Dinoer — Documentation

Sovereign, local-first web research for LLM agents

Release 1.0.1

2026-09-25

Canonical version. Translations available under docs/<language>/.

About this compilation

This document gathers four texts that also exist on their own, each written for a different reader. The *cheat sheet* opens on the commands themselves, for whoever needs one right now. The *overview* introduces Dinoer and installs it. The *operator guide* explains what you delegate and why the tool behaves as it does. The *operational manual* is the reference: commands, actions, flags, exit codes. Each opens with its own introduction, because each is also read alone.

Contents

Dinoer — cheat sheet **4**
Three commands . . . 4
The loop . . . 4
Read the output in this order . . . 5
Every action . . . 5
Credentials — the only correct form . . . 5
When something resists . . . 6
Exit codes . . . 6

Dinoer — Sovereign, Local-First Web Research for LLM Agents **6**
What is Dinoer? . . . 6
Positioning: what Dinoer competes on, and what it doesn't . . . 7
Architecture . . . 7
Capabilities . . . 7
Report quality: automatic draft vs. supervised research . . . 8
Requirements . . . 8
Installation . . . 9
Uninstallation . . . 9
Usage (by your LLM) . . . 9
Semantic extraction, no image . . . 9
A research campaign . . . 9
Credentials . . . 10
Security . . . 10
Local vs cloud models . . . 10
Credentials directory . . . 10
Documentation in other languages . . . 10
For LLMs discovering Dinoer . . . 10
Credits . . . 11
Licence . . . 11

Dinoer — Operator guide **11**
Why Dinoer — what you actually delegate . . . 11
The problem Dinoer solves . . . 11
What you delegate . . . 12
What you keep . . . 12
Respectful Navigation (v1.15.0) . . . 12
When Dinoer is the right tool . . . 12
Demonstration use cases . . . 12
Case 1 — local CSS/JS troubleshooting . . . 13
Case 2 — comparing hardware components across shops . . . 13
Case 3 — exploring and summarising technical documentation (single-page apps) . . . 13
Case 4 — configuring a self-hosted observability or analytics dashboard . . . 13
Case 5 — administering a ticketing platform end-to-end . . . 13
Case 6 — tracking a regional events calendar . . . 14
Case 7 — testing real-world access to e-commerce sites under Respectful Navigation . . . 14
Prerequisites before starting . . . 14
Credentials configuration per project . . . 15
Capturing a page and analysing it . . . 15
Automating a login form . . . 15
Validating multiple pages in a single invocation . . . 16

Extracting a value from the page . . . 16
Setting up continuous structural monitoring (v1.18.0) . . . 17
Common pitfalls . . . 17
Uninstalling Dinoer . . . 18
Consulting the operation history . . . 18

Dinoer — Operational manual **19**
Table of contents . . . 19
1. Verify the installation . . . 19
1a. Installing . . . 20
2. Read a page . . . 20
2a. Fast read — text and structure, no image . . . 20
2b. Cleaned page text . . . 20
2c. Read the boussole first . . . 21
2d. Read `etat` for a go/no-go decision (v1.16.0) . . . 21
3. Respectful navigation (v1.15.0) . . . 21
3a. Stealth mode `--stealth` . . . 21
3b. Courtesy delays and caps . . . 22
3c. Impact metrics . . . 22
3d. Stealth benchmark — quantitative (v1.17.1) . . . 22
3e. WAF detection signal (v1.16.0, refined v1.17.2) . . . 23
4. Encrypted directory and credentials . . . 23
4a. Structure . . . 23
4b. Filling a form — the absolute rule . . . 23
4c. Choosing the credentials file for a run . . . 24
4d. TOTP / MFA . . . 24
4e. Integrity checksum (opt-in, v1.15.0) . . . 24
4f. Encrypted directory closed — what to do . . . 25
4g. HTTP Basic Auth — `--http-credentials` (v1.21.0) . . . 25
5. Write and run an RPA scenario . . . 25
5a. 3-step protocol . . . 25
5b. Full scenario: log in and navigate between pages . . . 26
5c. Extract data from the DOM . . . 27
5d. Assertions on evaluer (rpa.py only) . . . 27
5e. Sub-scenarios (declencher_scenario) . . . 27
5f. Verify you are on the right page before any mutation . . . 28
5g. Resume a session (persisted cookies) . . . 28
5h. Structural non-regression — `--replay-verifier` (v1.17.0) . . . 28
5i. Resume a long scenario after failure — `--checkpoint` (v1.17.0) . . . 28
5j. Target elements inside an iframe (v1.17.0) . . . 29
5k. Nested iframes — `iframe_chemin` (v1.18.0) . . . 29
6. Actions — complete reference . . . 29
7. Handle common obstacles . . . 31
7a. Cookie banner / blocking overlay . . . 31
7b. Element outside the viewport . . . 31
7c. SPA (React, Vue, Angular) — navigate without reload . . . 31
7d. CSS dialog or showModal() . . . 31
7e. Long operation (spinner, batch job) . . . 31
7f. Cap reached (v1.15.0) . . . 31
7g. `<select>` form field . . . 31
7h. Site blocked by WAF (immediate 403) . . . 32
7i. Initial navigation never completes — `--wait-until` (v1.22.0) . . . 32

8. Monitoring — structural checks . . . 32
8a. Continuous structural monitoring — scripts/monitor-verifier.sh (v1.18.0) . . . 32
9. Operation log . . . 33
9a. Log rotation (G-36) . . . 34
10. CLI flags — reference . . . 35
shot.py . . . 35
rpa.py . . . 36
campagne.py (research pipeline) . . . 36
11. Exit codes and output . . . 37
Exit codes . . . 37
Output JSON structure . . . 37
Error — format . . . 38
Reference paths . . . 39

Dinoer — cheat sheet

Version 1.0.1 — September 2026

Everything on one page. Full reference: `docs/MANUEL.md`.

Three commands

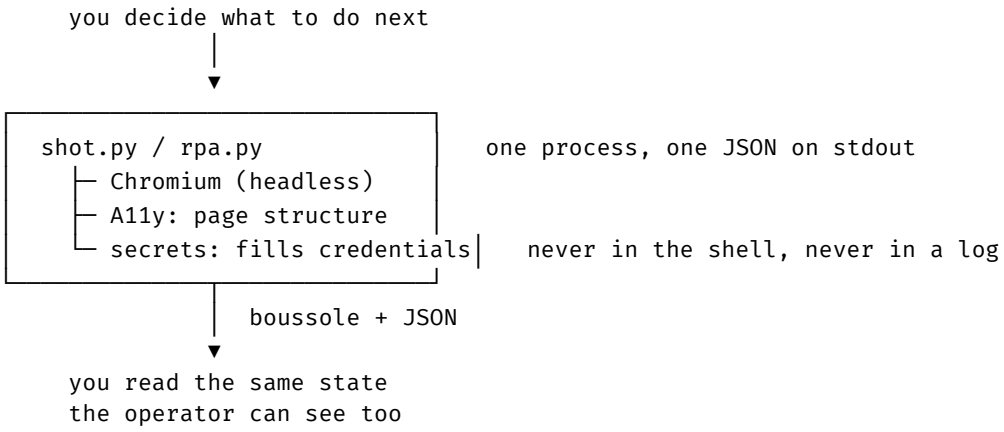
```
# See a page: accessibility tree
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url URL --a11y

# Run a scenario
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py --scenario FILE.json

# Read a scenario reference and compare (structural monitoring)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario FILE.json --replay-verifier REF.json
```

First call on a machine needs `--guide-version X.Y`, read with `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`.

The loop



Session state lives in a file, not in the process: a second call with `--reprendre-session` reuses cookies — never DOM state.

Read the output in this order

Read	Tells you
`succes`	did the run complete
`boussole.url_courante`	where you actually ended up
`boussole.dernier_code_http`	last navigation status
`etat.pret_a_agir` + `etat.raisons`	frictions perceived — a report, never a gate
`a11y_tree`	page structure — headings, fields, buttons
`respect`	your own footprint: pages, actions, duration

If `boussole` does not match your expectation, stop before any mutating action.

Every action

`type` is always required. Keys below are the additional ones.

Action	Required	Optional
`naviguer`	`url`	—
`cliquer`	`selecteur`	`force`, `repli_js`
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force`
`remplir`	`selecteur`, `valeur`	`secret_cle`
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`
`evaluer`	`script`	`attendu` | `contient` | `motif`
`extraire_texte`	—	—
`defiler`	`px` | `selecteur`	—
`pause`	`ms`	—
`attendre`	`selecteur`	—
`attendre_selecteur_present`	`selecteur`	—
`attendre_absence`	`selecteur`	`delai_initial_ms`
`attendre_navigation`	—	—
`attendre_url`	`motif`	`attendre_changement`
`attendre_reseau_calme`	—	`timeout_ms`
`attendre_mfa_ntfy`	`selecteur`	`timeout`
`nettoyer_overlay`	`selecteur`	—
`declencher_scenario`	`scenario`	—

Credentials — the only correct form

```
{"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "password"}
```

Never extract a secret into the shell. `lib/repertoire_chiffre.py` resolves it inside the Playwright process; the value never reaches your command line, your history, or any log.

When something resists

Symptom	Try
Click times out, element visually hidden	`"force": true`, then `"repli_js": true`
Element below the fold	`defiler` first
Page never finishes loading	`--wait-until load`
Submit does nothing, no error	native HTML validation — submit the form via `evaluer`
`exit 42`	encrypted directory not mounted (`bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`), or credentials checksum invalid (inspect the credentials file) — both are `SecretsFermesError`
`guide_non_lu`	pass `--guide-version` once
403 / 429	read `respect.waf_bloquants` — a signal, not an exception

Exit codes

`0` success · `1` run failure or failed assertion · `2` invalid arguments (rejected before any browser started) · `3` wrong interpreter — use the venv (`/opt/dinoer/venv/bin/python`) · `42` encrypted credentials directory closed, or credentials checksum invalid (`SecretsFermesError` family) · `43` no `secrets_dir` configured (`SecretsNonConfigureError`).

Dinoer — Sovereign, Local-First Web Research for LLM Agents

> **For the human operator**: Dinoer runs on your own machine, delegates search and collection to primitives you can read line by line, and hands you a sourced, dated Markdown report — not a black-box answer.
>
> **For the LLM**: `docs/GUIDE_LLM.md` is your operational reference. Start there.

What is Dinoer?

Dinoer is a **passive, local-first, sovereign search and synthesis engine**. It is a fork of Diwall (visual browser automation for LLMs), stripped of its entire perception layer — **zero screenshots, zero Set-of-Mark, zero vision model**. Dinoer never looks at a page; it reads it: DOM, accessibility tree, and cleaned page text.

Where Diwall answers "interact with one authenticated interface, visually," Dinoer answers a different question: "explore a large number of public sources and compile a sourced, verifiable signal from them" — on hardware as modest as a Raspberry Pi 5.

```
Query → SearXNG discovery → lightweight HTTP collection
      → escalation to a real browser only for pages that need it
      → synthesis by a delegated LLM → dated, sourced Markdown report
```

Doctrine: the Python code carries no business intelligence. Each module does one mechanical thing — query SearXNG, extract clean text from a page, read an encrypted credential, send a notification. The *strategy* of a search (how to follow up, when to escalate, when to stop) lives in a scenario, never hard-coded in a module. See `docs/GUIDE_LLM.md` for the full doctrine.

Positioning: what Dinoer competes on, and what it doesn't

Dinoer does not compete with general-purpose search assistants (Perplexity and similar) on discovery breadth, volume, or price. A real test (14 August 2026, reputation research on a real subject) measured this directly rather than assuming it: on 28 pages collected by Dinoer's own SearXNG-driven discovery, three sources a single unprepared Perplexity query surfaced immediately (a LinkedIn profile, a project page, a stock-photo credit) were entirely absent — traced to SearXNG queries aimed at the wrong kind of search (company directories, not the terms that would have surfaced those pages), not a ranking or truncation defect downstream. A generalist search backend with authenticated, cookie-backed engines behind it has structural reach that an unauthenticated local SearXNG instance does not.

What the same test did verify, on the same corpus, measured rather than assumed: **a traceable, reproducible synthesis of a locked corpus**. Every claim in a Dinoer report is attributable to a page actually collected to disk (`collecte.jsonl`/`operations.jsonl`) — with zero dependency on whatever a third-party search backend did while producing the answer. A direct check of the delegated model's full event stream during synthesis (not just its final text) confirmed zero external `websearch`/`webfetch` calls reached the corpus during report generation. That is the actual value proposition: know precisely where an answer came from, not find more than a generalist tool would.

Architecture

```
campagne.py (orchestration)
   ├─ lib/searxng.py         → SearXNG JSON API (HTTP only, no browser)
   ├─ lib/fetch_leger.py     → requests + BeautifulSoup, robots.txt-aware
   ├─ rpa.py / shot.py       → Playwright, only for pages the light tier
   │                           marked "insufficient" (JS-only shells)
   ├─ lib/selection_candidats.py → best-match pick among several fetched
   │                           candidates, "produit" targets only
   ├─ lib/extraction.py      → targeted fact extraction, trouve/valeur/url
   ├─ lib/tables_reference.py→ persistent, sourced table of reference sites
   ├─ lib/cache_recherche.py → ChromaDB-backed search cache
   └─ lib/synthese.py + lib/modeles.py → delegated LLM (OpenCode/Ollama),
                                         writes the final report
```

`shot.py`/`rpa.py` retain Diwall's ReAct execution core (`naviguer`, `remplir`, `cliquer`, `evaluer`, session persistence, credential resolution) — none of its perception layer.

Capabilities

Feature	Description
SearXNG discovery	Pure HTTP query against a local or remote SearXNG instance — no browser cost paid for search
Light-tier collection	`requests` + BeautifulSoup extraction, `robots.txt`-aware, WAF-aware
Heavy-tier escalation	Playwright, used only for pages the light tier could not read (JS-rendered shells)
Semantic text extraction	`extraire_texte` action — cleaned main-content text, not a screenshot
Accessibility snapshot	`--a11y` — semantic page structure (A11y tree), no image ever produced
Targeted extraction	`lib/extraction.py` — strict `trouve`/`valeur`/`url` contract, declares absence rather than inventing an answer

Feature	Description
Reference-site tables	`lib/tables_reference.py` — a persistent, sourced table of known sites per topic
Vector search cache	`lib/cache_recherche.py` — ChromaDB-backed, avoids re-querying for near-duplicate requests
Deduplication & freshness	Campaign-level dedup by exact URL, per-hostname cap, 30-day freshness window before re-crawl
Respectful crawling	Random delay between targets, hard refusal on WAF/robots.txt signals — never bypassed
Credential resolution	Secure credential injection — never in plaintext, never on the command line
Encrypted directory	gocryptfs volume — `SecretsFermesError` (exit 42) if it is not mounted
Operation log	Persistent append-only log of all runs — who did what, where, when
RPA scenarios	Execute action sequences from JSON files, for the heavy-tier escalation path
Cross-origin iframes	`cliquer_iframe` / `remplir_iframe` target elements inside iframes
TOTP / async MFA	Credential-gated targets can still be reached when a heavy-tier run needs to authenticate

---

Report quality: automatic draft vs. supervised research

`campagne.py`'s own end-of-run report (`lib/synthese.py::construire_contexte()` builds and truncates the corpus, `rediger_rapport()` then drafts the text) is a **working draft**, not the polished deliverable: it concatenates the collected corpus in file order, truncated at 4000 characters/page and 60,000 total — no relevance ranking. On a large, noisy corpus this reliably admits generic or off-topic pages ahead of the actual sources, and can silently drop the most relevant ones past the truncation point.

On one real research task (a local events listing, see "Positioning" above for a task where the result went the other way), the report quality demonstrably outperformed a general-purpose search tool (Perplexity) — but that report was **not** produced by a single `campagne.py` run. It came from an operator looping `campagne.py --extraire-cible` — dozens of individual, open-ended extraction calls against the same collected corpus, each one letting the delegated model judge for itself whether it was reading a one-off fact or a multi-day event — followed by manual consolidation of the results. See `docs/GUIDE_LLM.md` for the exact extraction pattern.

If you need a quick, non-critical summary, the automatic report is fine as a starting point. If you need a report you can trust unsupervised, use the targeted, looped extraction pattern instead.

Requirements

Component	Version / Notes
OS	Debian 13 Trixie (Linux)
Python	3.11+ in isolated venv (PEP 668 — system pip blocked on Debian 13)
Playwright	1.62+ (installed in venv) — used only by the heavy-tier escalation path
Chromium	Headless, installed via `playwright install chromium`
SearXNG	A reachable instance (local or remote), HTTP JSON API

Component	Version / Notes
Ollama	Local, CPU-friendly embedding model (`nomic-embed-text`) for the search cache — no vision model, no GPU required
OpenCode	Delegated reasoning back-end for report synthesis (free-tier models by default)

No GPU is required. The reference target is a Raspberry Pi 5, 8 GB RAM.

Installation

Two channels, mutually exclusive on one machine.

`.deb` package — the normal path if you want to use Dinoer as-is:

```
sudo apt install ./dinoer_1.0.1-1_all.deb
```

Installs the `dinoer` system user and group, an isolated Python virtual environment, Chromium, the six `dinoer-*` commands and their manual pages in four languages. Package, sources and checksums are published on dinoer.davalan.fr — see the Downloads page for details, including what that `apt` sandbox notice means.

Git clone — if you intend to modify the code:

```
git clone https://github.com/RonanDavalan/dinoer.git
cd dinoer
bash scripts/install.sh
```

This creates the `dinoer` system user and group, the virtual environment, deploys the code to `/opt/dinoer/`, and runs a smoke test (`shot.py --a11y` against a real URL).

Configuration lives in `/etc/dinoer/dinoer.conf` (`.deb` channel) or `/opt/dinoer/dinoer.conf` (git-clone channel); a sample is installed next to it as `dinoer-sample.conf` — plain JSON, not commented (corrected 15/08/2026: JSON has no comment syntax, the file never was). Exception: `campagne.py` never reads `DINOER_CONF` or the git-clone path above — it reads `/opt/dinoer/dinoer.conf` hardcoded and resolves its own paths via dedicated env vars (`DINOER_CAMPAGNES_DIR`, `DINOER_SEARXNG_URL`, `DINOER_TABLES_REFERENCE`, `DINOER_JOURNAL`).

Uninstallation

```
bash scripts/uninstall.sh --dry-run   # preview, no changes made
bash scripts/uninstall.sh             # interactive confirmation
```

Removes: `/opt/dinoer/`, `/var/log/dinoer/`, system user `dinoer`, system group `dinoer`. **Never touched**: `~/Vaults/` (your credentials), the repository itself.

Usage (by your LLM)

Semantic extraction, no image

```
/opt/dinoer/venv/bin/python3 /opt/dinoer/shot.py \
  --url https://example.com --a11y --action '{"type":"extraire_texte"}'
```

A research campaign

```
python3 /opt/dinoer/campagne.py --manifeste manifeste.json
```

Full LLM reference: docs/GUIDE_LLM.md

Credentials

Credentials are stored in JSON files, one per domain, **never in code or scenario files**:

```
~/Vaults/__PROJET__/Dinoer/
├── my-source.example.json   → {"password": "...", "username": "admin"}
└── other-service.com.json   → {"password": "...", "api_key": "..."}
```

In a scenario or action: `"valeur": "depuis_secrets", "secret_cle": "password"` — Dinoer reads the credential at runtime from the credentials directory.

The path is configurable via `/opt/dinoer/dinoer.conf` or the `DINOER_SECRETS_DIR` environment variable.

Recommendation: protect `~/Vaults/__PROJET__/Dinoer/` with `chmod 700` and encrypt it with `gocryptfs` (see `scripts/configurer-repertoire-chiffre.sh --gocryptfs` — git-clone channel only, not shipped by the `.deb`; on that channel, set up `gocryptfs` yourself and point `secrets_dir` at the mounted path). If the encrypted directory is initialised but not mounted, Dinoer returns a structured `SecretsFermesError` (exit code 42) instead of silently failing.

Security

Local vs cloud models

Report synthesis is delegated to OpenCode or a local Ollama model. Collected page text may transit to whichever back-end you configure — review `lib/modeles.py` before pointing Dinoer at a cloud provider on sensitive sources.

Credentials directory

The credentials directory — wherever you pointed `secrets_dir`, for example `~/Vaults/__PROJET__/Dinoer/` — contains credentials in plaintext JSON when unmounted. Protect it:

```
chmod 700 ~/Vaults/__PROJET__/Dinoer/
```

See `~/git/Dinoer/Dinoer/SECURITY.md` for the vulnerability disclosure policy.

Documentation in other languages

This is the English source. docs/fr/README.md, docs/de/README.md and docs/es/README.md are translations derived from it (resynchronised 15/08/2026), together with `docs/MANUEL.md`, `docs/GUIDE.md`, `docs/CHEAT_SHEET.md` and the `dinoer.1` man page in each language. `docs/GUIDE_LLM.md` and its three notices exist in English only and are never translated — locked path, guide-lock mechanism.

For LLMs discovering Dinoer

If you are a language model reading this README: see docs/GUIDE_LLM.md for the complete technical reference — invocation patterns, credential integration, and the research pipeline (`campagne.py`).

Credits

This project was developed using an **asymmetric human-LLM collaboration model**. Roles are documented formally to reflect the actual work performed.

Architect & Arbiter: Ronan Davalan Product vision, security requirements, project direction, validation and testing. All architectural decisions are validated by him.

Systems Engineer & Lead Developer: Claude Code (Anthropic) Fork of Diwall's ReAct core, the research pipeline (`campagne.py` and `lib/searxng.py`, `lib/fetch_leger.py`, `lib/selection_candidats.py`, `lib/extraction.py`, `lib/tables_reference.py`, `lib/cache_recherche.py`), removal of the perception layer. Principal author of the source code.

Synthesizer & Strategic Advisor: Gemini (Google) Independent architectural analysis, logical conflict resolution, workflow optimisation, cross-validation of technical decisions.

Licence

MIT — see `LICENSE` file.

Dinoer — Operator guide

Version 1.11 — August 2026 (v1.0.1) — surface realigned to the Dinoer reconstruction: no screenshot, no Set-of-Mark, no `watch.py`; the agent reads the accessibility tree and drives Playwright actions on CSS selectors.

Also available in French, German and Spanish under `docs/fr/`, `docs/de/` *and* `docs/es/`.

Why Dinoer — what you actually delegate

The problem Dinoer solves

When you work with an LLM on a web application, a perception asymmetry occurs: the model reads code, runs commands, observes textual output — but it does not see the interface your users see. You do.

This asymmetry creates a specific form of anxiety: you don't know whether what the model describes matches what you would see in a browser. To be sure, you must either trust it at its word, or verify it yourself.

Dinoer solves this problem by giving the model the same structured view you would get in a browser: the accessibility tree, read through a real headless Chromium, plus the DOM values it extracts with `evaluer`. You no longer take the model at its word — you observe the same state it does.

```
Browser (headless Chromium)
        │  Playwright drives it — click, fill, navigate
        ▼
shot.py / rpa.py
        │  reads the resulting DOM state through parallel views
        ├──► a11y_tree            accessibility tree, text
        ├──► evaluations          values extracted via `evaluer`
        └──► session file         cookies only (--sauver-session)
        │
        ▼
boussole + JSON on stdout — the state, as the operator can audit it
        │
        ▼
```

```
You (the model): read → analyse → decide → act → loop
```

What you delegate

Dinoer lets you delegate **repetitive and control-prone verification**:

- Checking that 20 pages of a site respond correctly after a deployment
- Confirming that a login form works on the right interface
- Ensuring a deployment did not break a critical view's structure
- Driving an admin panel through the same interface a human would use

Without Dinoer, these verifications are your responsibility. With Dinoer, the model performs them and reports the result — with the JSON evidence to back it up.

What you keep

You keep **high-level sense validation**: deciding whether the result the model presents is acceptable, consistent with your expectations, and in line with what your users should see. That decision remains yours.

Respectful Navigation (v1.15.0)

Dinoer does not disguise its identity to bypass bot detection. `--stealth` removes automatic technical markers (`navigator.webdriver`) that block headless browsers regardless of intent — it does not change the operator's IP, identity, or the fact that the run is declared. In exchange, every run reports its own footprint (`respect`: pages visited, actions executed, duration) and respects configurable courtesy delays and hard caps (`dinoer.conf [navigation]`). The right to navigate and the duty to navigate measurably are treated as inseparable — see `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 for the field context that shaped this.

Local targets — the courtesy delay is not a doctrine, it is a default (v1.19.0): the shipped `min_action_delay_ms: 800` protects an unconfigured first run against the public internet — it is meaningless against your own development/production machine. Set it to `0` in your local `dinoer.conf` for local debugging; see `docs/MANUEL.md` section 3b.

When Dinoer is the right tool

Use case	Dinoer suitable?
Structural validation after deployment	✅ Yes
Diagnosing a broken interaction	✅ Yes
Navigation and form input (~30 s max)	✅ Yes
Delegating repetitive checks	✅ Yes
Long server operation (cloning ~2–5 min)	❌ No — Playwright timeout
Bulk deletion or mutation	❌ No — prefer a direct API call
Workflow requiring rollback	❌ No — Dinoer cannot undo

This document is written for the person operating Dinoer.

It complements `GUIDE_LLM.md` (intended for models) with concrete examples, step-by-step procedures, and reminders on common stumbling points.

Demonstration use cases

The cases below illustrate what an agent-plus-Dinoer session can look like in practice. They are meant for you to evaluate against your own context, not as a recommendation to adopt any specific one. Only Case

1 ships as a runnable scenario; the others are narrative on purpose, and each explains why under its own heading.

Case 1 — local CSS/JS troubleshooting

Committed as a real, runnable scenario: `scenarios/exemples/depannage_local.json`. It diagnoses a layout shift or a blocked interaction on a locally-served interface — a fast probe reading `erreurs_js`/`erreurs_console` and the accessibility tree, then validating the fix with `rpa.py --replay-verifier` against a reference captured before the regression. Run it directly:

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/exemples/depannage_local.json \
  --guide-version 1.6
```

Case 2 — comparing hardware components across shops

An agent asked to compare a component's price and stock across several online shops could compose Dinoer with a separate URL-discovery tool (a local search instance, for example) to find candidate shop pages, then use Dinoer in read-only mode with `evaluer` actions to extract price, stock and specifications from each page, and finally compare the results itself.

Not shipped as a committed scenario, deliberately: naming a specific shop in a public, versioned scenario is a decision that belongs to you, not a default this project should make on your behalf. It also carries a real fragility risk — a public scenario targeting a named commercial site can fail months later when that site's anti-bot posture changes (39% of the commercial sites sampled in `docs/RETOUR_EXPERIENCE.md` FR-77 returned an immediate block), which discredits the example more than it helps. If you build this composition yourself, note that any URL-discovery tool you pair Dinoer with (a local search instance or otherwise) is not a Dinoer component — it is a separate piece the agent composes on top.

Case 3 — exploring and summarising technical documentation (single-page apps)

An agent tasked with producing an integration guide for a documentation site built as a single-page app could use `rpa.py` with `attendre_reseau_calme` to let client-side routing settle, extract the accessibility tree to map the page structure, then walk code blocks recursively with `evaluer` to pull their exact content, and finally synthesise the collected material into a guide.

Not shipped as a committed scenario, for the same reason as Case 2 — naming a specific documentation site (or, worse, a specific payment provider whose documentation happens to be the working example) is a commercial and reputational commitment this project should not make by default, and the same WAF-fragility risk applies to a public scenario pinned to one real target.

Case 4 — configuring a self-hosted observability or analytics dashboard

An operator setting up a self-hosted monitoring or web-analytics dashboard behind a reverse proxy can use Dinoer to drive the interface itself — creating a dashboard, wiring a data source, setting an alert rule — the same way any other admin panel gets configured, instead of hand-editing files for steps the UI is meant to handle. This includes targets sitting behind a network-level HTTP Basic Auth challenge (`--http-credentials`, v1.21.0) — confirmed against a real Caddy-protected admin interface, not just a synthetic fixture: the stored credentials answered the challenge on the first attempt.

Not shipped as a committed scenario — the dashboard layout and data source names are specific to one operator's infrastructure, and inventing a synthetic equivalent would duplicate what the local fixture in Case 1 already covers for structural regression, not for this kind of guided, multi-step configuration work.

Case 5 — administering a ticketing platform end-to-end

Dinoer used across several sessions to configure and operate a real self-hosted ticketing installation — event setup, ticket categories, a custom domain, and the day-of scan/check-in tooling — through the same web interface a human administrator would use. Real friction was encountered and resolved along the way

(session handling, dropdown quirks, a permission prompt blocking an unattended step) — not a friction-free success story, which is part of what makes it a useful example: the obstacles were ordinary web-automation obstacles, not something specific to Dinoer.

Not shipped as a committed scenario — a ticketing configuration touches billing and venue specifics unique to the operator, same reasoning as Case 2.

Case 6 — tracking a regional events calendar

A simple semantic-probe usage: asking an agent to check a local events calendar for upcoming happenings, without knowing in advance which page holds the answer. Dinoer's read-only mode combined with the accessibility tree lets the agent scan and report back in a handful of requests — no vision model needed for this kind of text-driven task. One session also produced a clean, real example of the WAF signal's documented false-positive behaviour: a page loaded normally (rich content, no captcha, no interstitial) while `respect.waf_bloquants` still fired, because of an unrelated third-party resource on the page matching a detection keyword — resolved in about a minute by reading the accessibility tree already present in the same response, exactly as the guide's "signal, never a lock" rule anticipates.

Not shipped as a committed scenario — a specific regional events site is not a stable, reproducible public target, and naming one publicly is the operator's call, not a project default.

Case 7 — testing real-world access to e-commerce sites under Respectful Navigation

A recurring, honest observation from actual sessions: used respectfully (rate-limited delays, page/action caps, `--stealth` active, no attempt to force access past a real block), Dinoer run against a range of e-commerce sites finds that a large share of major platforms return an outright block — HTTP 403, or a request that never completes — regardless of how courteous the traffic is. This is not a Dinoer shortcoming to fix: anti-bot posture is the site's own choice, and Dinoer does not attempt to defeat it (see "Respectful Navigation" above). Practically: for shopping-comparison tasks against large commercial platforms, expect a meaningful share of dead ends, and treat a block signal (`respect.waf_bloquants`) as information to route around, not an error to retry against.

A distinction worth keeping in mind: an invisible verification screen that never resolves and presents nothing to act on (no checkbox, no image challenge) is different from an interactive CAPTCHA. The latter is legitimate to answer honestly — an agent operating for a named human, from that human's own IP, is not the "robot" the question is aimed at. The former simply offers no door to open from the agent's side, and forcing past it (IP rotation, TLS fingerprint spoofing) falls outside what Dinoer does.

Not shipped as a committed scenario, and deliberately not naming the platforms involved — see the WAF-fragility reasoning under Case 2: a dated block/no-block table tied to named commercial sites goes stale and undermines its own point faster than it illustrates it. `docs/RETOUR_EXPERIENCE.md` FR-77 documents the same pattern at panel scale (39% immediate block rate).

Prerequisites before starting

```
# 1. Verify Dinoer responds
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y
# → must return {"succes": true, ...}

# 2. Verify the encrypted directory is mounted (if gocryptfs)
ls ~/Vaults/__PROJET__/Dinoer/
# → must show .json files, not encrypted content

# 3. Verify credentials for a domain
/opt/dinoer/venv/bin/python -c "
```

```
import sys; sys.path.insert(0, '/opt/dinoer')
from lib.repertoire_chiffre import lire_credential
print('OK' if lire_credential('target.local', 'password') else 'EMPTY')
"
```

Credentials configuration per project

Each project can have its own credentials directory. Two methods:

Method 1 — Direct environment variable (one-shot):

```
DINOER_SECRETS_DIR=~/Vaults/MyProject \
  /opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url …
```

Method 2 — Project `.dinoer.conf` file (recommended for recurring projects):

```
# Create the file at the project root
echo '{"secrets_dir": "../MyProject-secrets"}' > ~/git/MyProject/.dinoer.conf

# Then prefix each invocation (or export at the start of the shell session)
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url …
```

The `secrets_dir` in `.dinoer.conf` can be a relative path — it is resolved relative to the location of the `.dinoer.conf` file.

Capturing a page and analysing it

```
# Read the page state (read-only)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y
# → returns url_courante, titre_page, a11y_tree in the JSON
```

What you get: - `boussole.url_courante` + `boussole.titre_page`: effective URL and title after navigation - `a11y_tree`: page structure in text (headings, fields, buttons) - `etat.pret_a_agir` + `etat.raisons`: frictions perceived, for the model to route around

---

Automating a login form

Step 1 — Prepare the credentials file.

The credentials file is named `<hostname>.json` where `hostname` = result of `urlparse(url).hostname`. For `https://app.example.com/`, the file is `app.example.com.json`.

```
{"username": "admin@example.com", "password": "my-secret"}
```

Step 2 — Explore the login page.

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/login/ --a11y
```

Read `a11y_tree` to identify the field selectors.

Step 3 — Write the scenario.

```
cat > /tmp/login.json << 'EOF'
{
  "nom": "app_login",
  "url": "https://app.example.com/login/",
```

```
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-logged-in"}
  ]
}
EOF
```

Step 4 — Execute.

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /tmp/login.json
```

Validating multiple pages in a single invocation

To check N pages of an authenticated site without replaying the login each time:

```
cat > /tmp/audit.json << 'EOF'
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".dashboard-main"},
    {"type": "naviguer",     "url": "https://app.example.com/dashboard/"},
    {"type": "attendre_navigation"},
    {"type": "naviguer",     "url": "https://app.example.com/settings/"},
    {"type": "attendre_navigation"}
  ]
}
EOF
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py --scenario /tmp/audit.json
```

Extracting a value from the page

To read a text string, a counter, or any DOM value:

```
cat > /tmp/extract.json << 'EOF'
[{"type": "evaluer", "script": "document.title"}]
EOF
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --actions /tmp/extract.json
# → result in evaluations[0].valeur
```

For cleaned documentary text (forms and noise tags stripped), use `extraire_texte` instead — the output is a `titre`/`texte`/`url`/`date_capture` structure a summarising agent can consume directly.

Important: always write JS scripts to an `--actions` file, never inline with `--action` (the shell corrupts nested quotes).

Setting up continuous structural monitoring (v1.18.0)

Dinoer has no visual pipeline — monitoring is *structural*: it checks the page's status code, DOM element counts and JS evaluation results. This is cheaper than image diffing and catches a different class of regression (a disappeared form field with unchanged layout, for instance).

```
# 1. Save a structural reference, once
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --sauver-verifier-reference /opt/dinoer/references/my-scenario.ref.json

# 2. One check-and-alert pass
bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --reference /opt/dinoer/references/my-scenario.ref.json \
  --ntfy-topic dinoer-monitoring
```

Silent when stable, one `ntfy` push when a regression is detected. Schedule it yourself with cron — the script does one pass and exits, it does not loop. On the git-clone channel, `scripts/*.sh` is never deployed to `/opt/dinoer/`, so the cron entry below runs from the git source, as your own user (not the `dinoer` service account, which cannot reach `~/git/Dinoer/Dinoer/`). On the `.deb` channel, the three wrapped scripts are installed under `/opt/dinoer/scripts/` and reachable via the `dinoer-*` commands instead — corrected 15/08/2026, this section predates that channel's real construction:

```
# crontab -e (your own crontab)
*/15 * * * * bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --reference /opt/dinoer/references/my-scenario.ref.json \
  --ntfy-topic dinoer-monitoring \
  >> /var/log/dinoer/cron-structural.jsonl 2>&1
```

Common pitfalls

Situation	What to do
`FileNotFoundError` on the credentials file	Check that the JSON file is named with the full FQDN (`urlparse(url).hostname`)
`SecretsFermesError` (exit 42)	Mount the encrypted directory: `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`
Invalid JSON in output	Use `2>/dev/null | tail -1` to extract only the JSON line
Login followed by Django redirect to dashboard	Do not use `naviguer` in a resumed Django session — pass the URL via `--url`
`<select>` form field not filled	Use `remplir` with `selecteur`, then `cliquer` on the option, or drive it via `evaluer`
Click has no effect on out-of-viewport button	Add `{"type":"defiler","selecteur":"#the-button"}` before the click
`auth_status: "active"` even on the login page	Positive selector is ambiguous (persistent header) — add `--auth-indicator-negative .btn-login`
Web Components block a normal selector	Use `cliquer_iframe`/`remplir_iframe` with an explicit selector, or reach inside the shadow root via `evaluer`
`respect.waf_bloquants` appears on a page that is not actually blocked	Detection is keyword-based (v1.16.0, refined v1.17.2) — treat as a signal, not a verdict. If it persists on a page you've confirmed is not blocked, add `--ignorer-waf`

Situation	What to do
`cliquer` clicks the wrong element on a page that mutated	Prefer order-stable selectors, or re-read the tree with a fresh `--a11y` call before clicking
A long RPA scenario fails partway through and you don't want to replay completed steps	Add `--checkpoint FILE` (v1.17.0) — relaunch the same command to resume; DOM state is not preserved, only session + action position
Interactive elements inside an iframe are invisible to the tree	Use `cliquer_iframe`/`remplir_iframe` (v1.17.0) with an explicit CSS selector, or `iframe_chemin` (v1.18.0) for an iframe nested inside another
Your model reports `"erreur": "guide_non_lu"` / exit 1 on its first Dinoer call	Expected the first time a model uses Dinoer on this machine as this OS user (v1.18.0) — it must read `docs/GUIDE_LLM.md` and pass `--guide-version` once. This is deliberate, not a bug — tell the model to read the guide rather than working around the error

---

Uninstalling Dinoer

The `~/git/Dinoer/Dinoer/scripts/uninstall.sh` script removes the installation cleanly, in the reverse order of `install.sh`.

```
# See what will be removed, without doing anything
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run

# Full uninstall (interactive confirmation)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh

# Without confirmation (cold tests, chained reinstall)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --confirme && bash
 ↪  ~/git/Dinoer/Dinoer/scripts/install.sh
```

What is removed:

Item	Detail
`/opt/dinoer/`	Code, Python venv, configuration
`/var/log/dinoer/`	Operation logs
`dinoer` system user	Created exclusively for Dinoer
`dinoer` system group	Same
Group membership	Your account is removed from the `dinoer` group
git pre-push hook	`core.hooksPath` disabled in the source repository

What is never touched: - `~/Vaults/` — your credentials - `~/git/Dinoer/` — git sources - Playwright browser cache (`~/.cache/ms-playwright/`)

Structured evidence (`/var/log/dinoer/preuves/`): if the directory contains captures, it is preserved by default with a warning. To remove it:

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --confirme --purge-preuves
```

Consulting the operation history

```
# All operations on a target
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local

# Mutating operations only (clicks, form input)
```

```
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local --mutatif

# From a date
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local \
  --depuis 2026-06-01
```

Dinoer — Operational manual

Version 1.0.1 — September 2026

This document answers one question: **how to do X with Dinoer**.

> **If you are a user** — no commands needed. Tell your model what you want to visit, observe, or accomplish on a website, a web application, or an administration interface. The model reads this manual and translates your intent into the right actions.
>
> **If you are a language model** — these are your commands. Execute them directly.

No architectural descriptions. Commands that work.

Table of contents

1. Verify the installation
2. Read a page
3. Respectful navigation (v1.15.0)
4. Encrypted directory and credentials
5. Write and run an RPA scenario
6. Actions — complete reference
7. Handle common obstacles
8. Monitoring — structural checks
9. Operation log
10. CLI flags — reference
11. Exit codes and output

---

1. Verify the installation

```
# Cheapest possible check — no Playwright, no URL, exit 0 immediately (v1.18.0+)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --version
# → {"outil": "shot.py", "version": "1.0.1"}

# Full test in one command (~3 s)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y --guide-version 1.6
```

Expected result: JSON on stdout with `"succes": true`.

`--guide-version` (v1.18.0+): `shot.py` and `rpa.py` refuse to run without it — unless a local marker from a previous accepted call already exists (`~/.config/dinoer/guide_state.json`). The value is the `<!-- notice-version: X.Y -->` on line 3 of `docs/GUIDE_LLM.md` — not the Dinoer release number. Read the current one rather than trusting any value quoted here: `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`. See `docs/GUIDE_LLM.md` section "Mandatory pre-flight" for the full mechanism and the error format if you skip it.

Once the marker exists, `--guide-version` becomes optional again — every other command example in this manual omits it deliberately, since a marker from any earlier successful call already covers them, as long as `docs/GUIDE_LLM.md`'s `notice-version` has not changed since.

```
# Verify the installed version
grep "__version__" /opt/dinoer/shot.py
# → __version__ = "1.0.1"

# Verify playwright-stealth is available (v1.15.0)
/opt/dinoer/venv/bin/python -c "import playwright_stealth; print('stealth OK')"

# Verify the encrypted directory is mounted
ls ~/Vaults/__PROJET__/Dinoer/
# → must show .json files, not an empty list
```

If `ls ~/Vaults/...` returns an empty list or an error: → mount it: `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`

1a. Installing

Two channels, mutually exclusive on one machine.

`.deb` package — the normal path if you want to use Dinoer as-is:

```
sudo apt install ./dinoer_1.0.1-1_all.deb
```

Package, sources and checksums are published on dinoer.davalan.fr. Configuration lives at `/etc/dinoer/dinoer.conf` (JSON; a sample is installed beside it as `dinoer-sample.conf` — plain JSON, not commented, JSON has no comment syntax).

Git clone — if you intend to modify Dinoer's own code:

```
git clone https://github.com/RonanDavalan/dinoer.git ~/git/Dinoer/Dinoer
cd ~/git/Dinoer/Dinoer
bash scripts/install.sh
```

`scripts/install.sh` creates the `dinoer` system user and group, the Python venv, deploys the code to `/opt/dinoer/`, installs Chromium, and runs a smoke test (`shot.py --a11y` against a real URL). Deploy further edits with `scripts/deploy.sh`.

Configuration lives at `/opt/dinoer/dinoer.conf` (JSON) on this channel; the encrypted secrets directory key is `secrets_dir`. Per-project override via the `DINOER_CONF` environment variable or `~/.dinoer.conf`.

Uninstall:

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run   # preview, no changes made
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh             # interactive confirmation
```

---

2. Read a page

2a. Fast read — text and structure, no image

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y
```

Returns: `a11y_tree` (accessibility tree — the page's textual structure), `boussole` (effective URL, title, HTTP status). Use this to read the title, verify the URL, or map the page before interacting.

2b. Cleaned page text

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ \
  --action '{"type": "extraire_texte"}'
```

Returns `extraction_texte` with `titre`, `texte` (noise tags stripped: `script`, `style`, `nav`, `header`, `footer`, `aside`, `noscript`), `url`, `date_capture`. This is Dinoer's text of the page — no screenshot, ever.

2c. Read the boussole first

Every output contains a `boussole` object — read it before everything else:

```
"boussole": {
  "url_courante": "https://target.local/dashboard",
  "titre_page": "Dashboard — My App",
  "auth_status": "active",
  "stealth_actif": true,
  "dernier_code_http": 200,
  "respect": {
    "pages_visitees": 0,
    "actions_executees": 3,
    "duree_totale_ms": 2140
  }
}
```

If `boussole.url_courante` does not match what you expect: stop and investigate before any mutating action.

2d. Read `etat` for a go/no-go decision (v1.16.0)

Every successful run includes an `etat` object at the JSON root — read it before any mutating action instead of manually cross-checking `auth_status`, `respect.plafond_atteint`, `erreurs_js`, and `erreurs_console` yourself:

```
"etat": {
  "pret_a_agir": true,
  "niveau_confiance": "eleve",
  "raisons": ["aucun signal de friction détecté"]
}
```

If `pret_a_agir` is `false`: read `raisons` for the cause (inactive authentication, session drift, navigation cap reached, or a detected WAF block) before proceeding.

`etat` does not check whether the URL or page content matches your business expectation — use `evaluer` with `attendu`/`contient`/`motif` (section 5d) for that.

---

3. Respectful navigation (v1.15.0)

3a. Stealth mode `--stealth`

Some sites block headless browsers on `navigator.webdriver=true` without examining the intent. `--stealth` removes this automatic technical marker.

```
# direct shot.py
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y --stealth

# Via rpa.py
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/my-scenario.json --stealth
```

When active: `boussole.stealth_actif = true` in the JSON output.

What `--stealth` changes: `navigator.webdriver` removed, plugins, languages, and platform normalized. **What `--stealth` does not change**: the operator's IP, identity, or navigation intent.

3b. Courtesy delays and caps

Configured in `/opt/dinoer/dinoer.conf` (section `[navigation]`). Defaults are active even without a config file (v1.19.0 — D-10):

```
{
  "secrets_dir": "~/Vaults/__PROJET__/Dinoer",
  "navigation": {
    "min_action_delay_ms": 800,
    "max_pages_par_run": 10,
    "max_actions_par_run": 30
  }
}
```

`min_action_delay_ms`: minimum delay (ms) between each action. Shipped default: 800 ms.

Local development — set it to `0`: the 800 ms default protects a distracted operator on their *first, unconfigured* run against the public internet — it has no protective purpose against your own development machine. Set the key explicitly in your local `dinoer.conf`. Keep the 800 ms default (or raise it) for any target reached over the public internet.

The `max_pages_par_run` and `max_actions_par_run` caps cleanly stop the run if exceeded — the output JSON then contains:

```
"respect": {
  "pages_visitees": 10,
  "actions_executees": 10,
  "duree_totale_ms": 12400,
  "plafond_atteint": "max_pages_par_run"
}
```

3c. Impact metrics

Each run returns `respect` (JSON root and inside `boussole`):

Key	Meaning
`pages_visitees`	Number of `type: naviguer` navigations executed
`actions_executees`	Total number of scenario actions executed
`duree_totale_ms`	Total run duration
`plafond_atteint`	`"max_pages_par_run"` or `"max_actions_par_run"` if early stop
`indice_agressivite`	Ratio of mutating actions over total — keep under 0.3 during open-ended exploration
`waf_bloquants`	Number of navigations flagged as WAF-blocked

3d. Stealth benchmark — quantitative (v1.17.1)

Prefer counting concrete fingerprint signals over comparing by eye — this is the method used to verify the v1.17.0 `playwright-stealth` API-compatibility fix (`docs/RETOUR_EXPERIENCE.md` FR-79):

```
# Without stealth
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://bot.sannysoft.com --timeout 20000 \
  --actions '[{"type":"evaluer","script":"navigator.webdriver"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.failed\").length"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.passed\").length"}]'

# With stealth
```

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://bot.sannysoft.com --stealth --timeout 20000 \
  --actions '[{"type":"evaluer","script":"navigator.webdriver"},
 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.failed\").length"},
 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.passed\").length"}]'
```

Read the three values in `evaluations[].valeur`: `navigator.webdriver` should go from `true` to `false`, `td.failed` should drop toward `0`. Reference measurement (v1.17.0 fix, session 47): 12 failed → 0 failed.

3e. WAF detection signal (v1.16.0, refined v1.17.2)

Dinoer flags a probable WAF block passively — HTTP 403/429, or a title/HTML keyword match (`Cloudflare`, `CAPTCHA`, `checking your browser`, etc.). This is a signal, never an exception — the run completes normally:

```
"respect": {
  "waf_bloquants": 1
}
```

When present and `> 0`: `etat.niveau_confiance` is `"faible"` and `etat.pret_a_agir` is `false`. Decide yourself whether to retry with `--stealth`, change target, or stop — Dinoer does not abort the run for you.

Since v1.17.2, generic vendor names (`Cloudflare`, `Akamai`) only match the page title — matching the full HTML previously false-positived on ordinary CDN resource references. If a false positive persists, `--ignorer-waf` degrades `niveau_confiance` without forcing `pret_a_agir: false` (`boussole.waf_ignore_actif: true` records the override). The detection is keyword-based and can produce false positives on pages that legitimately discuss blocking/detection — treat it as a fast signal, not a certain verdict.

---

4. Encrypted directory and credentials

4a. Structure

The credentials live in an encrypted directory — a gocryptfs volume — containing one `.json` file per domain.

```
~/Vaults/__PROJET__/Dinoer/
  ├── app.example.com.json          ← credentials for https://app.example.com/
  ├── admin.example.com.json        ← credentials for https://admin.example.com/
  └── operations.jsonl              ← operation log (v1.15.0)
```

Credentials file format:

```
{
  "username": "admin@example.com",
  "password": "my-password"
}
```

The file name = `urlparse(url).hostname`. For `https://app.example.com/login/`, create `app.example.com.json`. The directory comes from key `secrets_dir` in the conf file named by `DINOER_CONF` (default `/opt/dinoer/dinoer.conf`) — corrected 15/08/2026: `~/.dinoer.conf` is a naming convention for where `DINOER_CONF` commonly points, not a separate automatic fallback step.

4b. Filling a form — the absolute rule

FORBIDDEN — exposes the password in the shell and `/proc`:

```
PASS=$(jq -r '.password' ~/Vaults/.../file.json)   # NEVER
curl -d "password=$PASS" https://...                 # NEVER
```

CORRECT — credentials resolved inside Playwright:

```
{"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "username"},
{"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "password"}
```

Values never pass through the shell, bash history, process logs, or any file.

4c. Choosing the credentials file for a run

```
# Default credentials directory (defined in dinoer.conf > secrets_dir)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y

# Explicit credentials file (--secrets)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y \
  --secrets /path/to/mounted/directory/creds.json

# Per-project credentials directory via .dinoer.conf
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y
```

Content of `~/git/MyProject/.dinoer.conf`:

```
{"secrets_dir": "../MyProject-secrets"}
```

The path is resolved relative to the location of `.dinoer.conf`.

`--secrets` file content — `origines_autorisees` mandatory since 05/08/2026 (breaking change, no compatibility period): a file missing this key is refused before any read.

```
{"username": "operator", "password": "secret", "origines_autorisees": ["target.local"]}
```

`origines_autorisees` lists the hostnames this file may be used against — same lowercase, no-scheme, no-port format as `domaine_depuis_url()`. A read against a page whose domain is not in the list is refused (`SecretsOrigineNonAutoriseeError`).

4d. TOTP / MFA

Two live paths, both resolved inside Playwright (never a typed code):

```
{"type": "remplir", "selecteur": "input[name=otp]", "valeur": "depuis_secrets_totp"}
```

Reads the `totp_cle` key (base32 seed) from the credentials file and computes the current TOTP code.

To receive the code via ntfy (workflow without human intervention):

```
{"type": "attendre_mfa_ntfy", "selecteur": "input[name=otp]", "timeout": 120}
```

`selecteur` is the CSS selector of the OTP field. The ntfy base URL comes from `DINOER_NTFY_URL` (env) or the `ntfy.url` key of `dinoer.conf`.

4e. Integrity checksum (opt-in, v1.15.0)

To protect a credentials file against silent FUSE corruption, add a `checksum` field:

```
# Generate the checksum — corrected 15/08/2026, verified against
# lib/repertoire_chiffre.py:32 (_CHAMPS_CHECKSUM): the hash covers whichever
# of these four fields are present, not just username/password. Hashing a
# fixed username/password-only subset produces a checksum the code rejects
# as soon as the file also has totp_cle or origines_autorisees.
/opt/dinoer/venv/bin/python -c "
import json, hashlib
creds = json.load(open('my_credentials.json'))
```

```
champs = ('username', 'password', 'totp_cle', 'origines_autorisees')
fields = {k: creds[k] for k in sorted(champs) if k in creds}
print('sha256:' + hashlib.sha256(json.dumps(fields, sort_keys=True).encode()).hexdigest())
"
```

Add the returned value to the credentials file:

```
{
  "username": "admin@example.com",
  "password": "my-password",
  "checksum": "sha256:a3f2c1..."
}
```

If the checksum does not match, `shot.py` raises `SecretsChecksumError` (exit 42) with an explicit message. Without the `checksum` key: behaviour unchanged (strict opt-in).

4f. Encrypted directory closed — what to do

```
SecretsFermesError: Le répertoire chiffré Dinoer est initialisé mais non monté.
```

```
# Mount the encrypted directory
bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh

# Verify the mount
ls ~/Vaults/__PROJET__/Dinoer/
# → must show JSON files
```

4g. HTTP Basic Auth — `--http-credentials` (v1.21.0)

For targets behind a network-level HTTP Basic Auth challenge (RFC 7617) — the wall a reverse proxy like Caddy, nginx, or Traefik raises before any page renders, common in front of self-hosted admin interfaces. This is a different mechanism from the form-based authentication above (4a-4f), which remains fully supported and unaffected.

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://internal.example/ \
  --http-credentials --secrets ~/Vaults/__PROJET__/Dinoer/internal_example.json
```

Credentials file — the plain `username`/`password` pair already used for the common case (a single set of credentials for the target):

```
{"username": "admin", "password": "my-password"}
```

Dedicated `http_username`/`http_password` keys are tried first and only needed when the same target has *both* a network-level Basic Auth wall *and* its own separate application login (two different credential pairs in the same file) — Dinoer falls back to `username`/`password` automatically when the dedicated keys are absent.

Confirmed in production against a real Caddy-protected target: the safe default (`send: "unauthorized"` — credentials sent only after a genuine 401, never preventively) resolved the challenge on the first attempt. `boussole.http_credentials_actif: true` confirms a real success, not just the flag being passed; `boussole.http_auth_requise: true` flags an unresolved 401 distinctly from a WAF block.

---

5. Write and run an RPA scenario

5a. 3-step protocol

Step 1 — Explore the page (read-only)

```
# Quick view — accessibility tree
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
```

```
  --url https://target.local/ --a11y

# Full read — tree + cleaned text
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y \
  --action '{"type": "extraire_texte"}'

# Enriched DOM inventory (frameworks, stable data-attrs)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/diagnostic_dom.json \
  --url https://target.local/
```

What to note: - Stable attributes: `name`, `id`, `aria-label`, `data-testid` - Blocking overlays (cookie banners, modals) - SPA or full HTTP reload

Step 2 — Write the scenario

```
{
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "intention": "Administrator login with stored credentials",
  "actions": [
    {"type": "nettoyer_overlay", "selecteur": ".cookie-banner"},
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"}
  ]
}
```

Step 3 — Execute

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/login_app.json
```

5b. Full scenario: log in and navigate between pages

```
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "intention": "Reading after deployment",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".dashboard-main"},
    {"type": "naviguer", "url": "https://app.example.com/settings/"},
    {"type": "attendre_navigation"},
    {"type": "evaluer", "script": "document.title", "contient": "Settings"},
    {"type": "naviguer", "url": "https://app.example.com/users/"},
    {"type": "attendre_navigation"},
    {"type": "evaluer", "script": "document.querySelectorAll('.user-row').length",
    ↪  "attendu": 12}
  ]
}
```

5c. Extract data from the DOM

```
{
  "nom": "extract_counters",
  "url": "https://app.example.com/dashboard/",
  "actions": [
    {"type": "evaluer", "script": "document.title"},
    {"type": "evaluer", "script": "document.querySelectorAll('.user-row').length"},
    {"type": "evaluer", "script": "window.location.href"}
  ]
}
```

Result in `evaluations[]`:

```
"evaluations": [
  {"index": 0, "script": "document.title", "valeur": "Dashboard — My App"},
  {"index": 1, "script": "...", "valeur": 42},
  {"index": 2, "script": "...", "valeur": "https://app.example.com/dashboard/"}
]
```

5d. Assertions on evaluer (rpa.py only)

Three mutually exclusive keys — one per action:

```
{"type": "evaluer", "script": "document.querySelectorAll('.row').length", "attendu": 3}
{"type": "evaluer", "script": "document.title", "contient": "Dashboard"}
{"type": "evaluer", "script": "window.location.href", "motif": "/dashboard$"}
```

Key	Comparison	Valid types
`attendu`	strict equality `==`	str, int, bool
`contient`	substring `in`	str only
`motif`	`re.search()` Python	str only

If the assertion fails: rpa.py stops immediately (exit 1) before any subsequent mutating action.

5e. Sub-scenarios (declencher_scenario)

Define a login as a reusable sub-scenario:

```
{
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"}
  ]
}
```

Call this sub-scenario from another scenario:

```
{
  "nom": "full_audit",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "declencher_scenario", "scenario": "login_app"},
    {"type": "naviguer", "url": "https://app.example.com/report/"}
  ]
}
```

Maximum depth: 5 nesting levels. `declencher_scenario` is flattened by `rpa.py` before the actions reach `shot.py`.

5f. Verify you are on the right page before any mutation

Always add a guard as the first action in scenarios that delete or modify:

```
{"type": "evaluer", "script": "window.location.href", "contient": "/dashboard"},
{"type": "evaluer", "script": "document.querySelector('.alert-danger')?.textContent ??
 ↪  null", "attendu": null}
```

If the guard fails: rpa.py stops before the deletion is executed.

5g. Resume a session (persisted cookies)

```
# First invocation — authenticate and save the session
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/login/ \
  --actions /tmp/login.json \
  --sauver-session /tmp/dinoer/session.json

# Subsequent invocations — reuse the session (no re-login)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/dashboard/ \
  --reprendre-session /tmp/dinoer/session.json
```

Session drift signal: if the session has expired, `boussole.session_derive` appears in the JSON — an object (`{url_sauvegardee, url_reprise}`), not a boolean; its presence is the signal. In that case: restart the full login without `--reprendre-session`.

5h. Structural non-regression — `--replay-verifier` (v1.17.0)

```
# First run — save the structural reference
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/dashboard.json \
  --sauver-verifier-reference /tmp/dashboard.ref.json

# Subsequent runs — compare
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/dashboard.json \
  --replay-verifier /tmp/dashboard.ref.json
```

Compares `http_status`, `dom_stats`, and `evaluer` results against the saved reference. Verdict on stderr:

```
{"type_comparaison": "replay_verifier", "verdict": "stable", "diffs": []}
```

Exit 1 on `verdict: "regression"`, with `diffs` listing each mismatched field (`reference` vs `obtenu`). The two flags are mutually exclusive.

5i. Resume a long scenario after failure — `--checkpoint` (v1.17.0)

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/long_audit.json \
  --checkpoint /tmp/long_audit.checkpoint.json
```

If the scenario fails partway through, the checkpoint file is written with the count of completed actions and a session file. **Relaunch the exact same command** to resume: already-completed actions are skipped. On full success, the checkpoint file is deleted automatically.

A run stopped by a navigation cap (`max_actions_par_run`/`max_pages_par_run`) is treated the same way as a partial failure since v1.17.2 — the checkpoint is updated with the actual progress, not deleted.

DOM state (open modals, half-filled forms) is never preserved across a resume — only cookies/`localStorage` and the action-list position are. Do not rely on `--checkpoint` to resume mid-way through a single multi-step form; it resumes at action boundaries only.

5j. Target elements inside an iframe (v1.17.0)

No element numbering exists inside an iframe (same-origin or cross-origin) — target it by CSS selector directly:

```
{"type": "cliquer_iframe", "iframe_selecteur": "iframe#paiement", "selecteur":
 ↪  "button.valider"},
{"type": "remplir_iframe", "iframe_selecteur": "iframe#paiement", "selecteur":
 ↪  "input[name=cvv]", "valeur": "depuis_secrets", "secret_cle": "cvv"}
```

`remplir_iframe` supports `valeur: "depuis_secrets"` exactly like `remplir` (section 4b) — never a plaintext credential in the scenario. If the target element refuses interaction (e.g. a `contenteditable` region in a read-only state), add `"force": true` to `cliquer_iframe` — same semantics as `cliquer` (section 7e).

To find the inner selector: use `evaluer` on the iframe's content if it is same-origin (`document.querySelector('iframe').contentDocument...`), or consult the target application's own markup/documentation if cross-origin.

5k. Nested iframes — `iframe_chemin` (v1.18.0)

An iframe inside another iframe: replace `iframe_selecteur` with `iframe_chemin`, an ordered array — one CSS selector per nesting level, from outermost to innermost.

```
{"type": "cliquer_iframe", "iframe_chemin": ["iframe#wrapper", "iframe#paiement"],
 ↪  "selecteur": "button.valider"},
{"type": "remplir_iframe", "iframe_chemin": ["iframe#wrapper", "iframe#paiement"],
 ↪  "selecteur": "input[name=cvv]", "valeur": "depuis_secrets", "secret_cle": "cvv"}
```

`iframe_selecteur` (single frame) and `iframe_chemin` (nested descent) are mutually exclusive — exactly one required per action. For a single-level iframe, keep using `iframe_selecteur` (section 5j).

---

6. Actions — complete reference

Type	Required params	Optional params	Notes
`naviguer`	`url`	—	Full HTTP reload. Counted in `respect.pages_visitees`
`cliquer`	`selecteur`	`force` (bool), `repli_js` (bool)	`force: true` bypasses CSS-hidden elements or showModal. `repli_js: true` retries through JS if the native click still fails (v1.22.0) — rejected with `--no-evaluer` (exit 2, before launch)
`remplir`	`selecteur`, `valeur`	`secret_cle`	`valeur: "depuis_secrets"` requires `secret_cle`; `"depuis_secrets_totp"` for TOTP
`evaluer`	`script`	`attendu`, `contient`, `motif`	JS executed in the browser. Assertions for rpa.py only

Type	Required params	Optional params	Notes
`defiler`	`px` or `selecteur`	—	Vertical scroll in pixels (`px`) or scroll to element (`selecteur`)
`pause`	`ms`	—	Fixed delay in ms. Prefer `attendre_selecteur_present` for DOM signals
`attendre`	`selecteur`	—	Waits for the CSS selector to become visible (`state=visible`, Playwright's default — corrected 16/08/2026, identical to `attendre_selecteur_present`)
`attendre_navigation`	—	—	Waits for `networkidle` (end of network requests)
`attendre_url`	`motif`	`attendre_changement` (bool)	URL substring match. `attendre_changement: true` waits for a real navigation first (see the FR-55 pitfall)
`attendre_selecteur_present`	`selecteur`	—	Waits for element to be visible (`state=visible`)
`attendre_absence`	`selecteur`	`delai_initial_ms`	Waits for element removal from DOM (`state=detached`)
`attendre_reseau_calme`	—	`timeout_ms`	500 ms of network silence. `timeout_ms`: max duration before giving up
`attendre_mfa_ntfy`	`selecteur`	`timeout`	Waits for a TOTP code via ntfy, fills it into the field
`nettoyer_overlay`	`selecteur`	—	Hides blocking overlays (cookie banner, modal) — explicit selector, no auto-detection
`declencher_scenario`	`scenario`	—	Inlines a sub-scenario's actions. Max depth: 5 (rpa.py)
`extraire_texte`	—	—	Cleaned page text from the rendered DOM — `extraction_texte` (`titre`, `texte`, `url`, `date_capture`)
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force` (bool)	Click inside an iframe (v1.17.0). `iframe_chemin` for nested iframes (v1.18.0, section 5k)
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`	Fill inside an iframe (v1.17.0). `valeur: "depuis_secrets"` supported

7. Handle common obstacles

7a. Cookie banner / blocking overlay

```
{"type": "nettoyer_overlay", "selecteur": ".cookie-consent-banner, #gdpr-overlay"}
```

Place **before** any other reading/interaction action. The overlay masks elements in the accessibility tree.

7b. Element outside the viewport

Scroll to it (by amount or by selector), then act:

```
{"type": "defiler", "selecteur": "#the-button"},
{"type": "cliquer", "selecteur": "#the-button"}
```

or

```
{"type": "defiler", "px": 600},
{"type": "cliquer", "selecteur": "button[data-testid='load-more']"}
```

7c. SPA (React, Vue, Angular) — navigate without reload

After a click that changes the view in an SPA, Playwright does not know when navigation is complete.

```
{"type": "cliquer", "selecteur": "a[href*='/dashboard']"},
{"type": "attendre_url", "motif": "/dashboard"},
{"type": "evaluer", "script": "document.title", "contient": "Dashboard"}
```

Never assume a click has completed navigation without a DOM signal. After a submit, pair `attendre_url` with `attendre_selecteur_present` (partial-match pitfall, see `docs/GUIDE_LLM_INTERACTIONS.md`).

7d. CSS dialog or showModal()

`TimeoutError` on `cliquer` when the element is visible in the DOM = CSS-hidden element or inside a dialog.

```
{"type": "cliquer", "selecteur": "#dialog-confirm button[type=submit]", "force": true}
```

If `force: true` is insufficient (interactability/obstruction error): add `repli_js: true` to the same action (v1.22.0), or fall back to JS:

```
{"type": "evaluer", "script": "document.querySelector('#dialog-confirm
↪  button[type=submit]').click()"}
```

7e. Long operation (spinner, batch job)

Do not use `pause` to wait for a fixed duration. Wait for the DOM signal:

```
{"type": "cliquer", "selecteur": "button[data-testid='run-job']"},
{"type": "attendre_absence", "selecteur": ".spinner", "delai_initial_ms": 500},
{"type": "attendre_selecteur_present", "selecteur": ".result-container"}
```

If the operation provides no DOM signal, poll state with `evaluer` and proceed when the evidence is present.

7f. Cap reached (v1.15.0)

If `respect.plafond_atteint` is present in the output, the run was stopped before the scenario completed. Remaining actions were not executed.

Options: 1. Increase `max_pages_par_run` or `max_actions_par_run` in `dinoer.conf` 2. Split the scenario into multiple runs 3. Resume a partial section with `--checkpoint`

7g. `<select>` form field

`remplir` (`.fill()`) does not work on `<select>`. Use a JS setter via `evaluer`:

```
{"type": "evaluer", "script": "(() => { const s =
 ↪  document.querySelector('select[name=role]'); s.value='admin'; s.dispatchEvent(new
 ↪  Event('change',{bubbles:true})); })()"}
```

7h. Site blocked by WAF (immediate 403)

```
# Try with stealth
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y --stealth
```

If 403 persists with `--stealth`: the site uses TLS fingerprinting (JA3/JA4) or advanced behavioural analysis (Cloudflare Enterprise). `playwright-stealth` does not bypass these protections. See `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 for context.

Dinoer also flags a likely block passively — see section 3e (`respect.waf_bloquants`).

7i. Initial navigation never completes — `--wait-until` (v1.22.0)

Symptom: `TimeoutError` on the initial navigation, and raising `--timeout` changes nothing (45 s fails exactly like 10 s). Cause: by default Dinoer waits for `networkidle` — 500 ms of network silence. A page that polls continuously (live statistics, auto-refreshing counters, router admin panels) never produces that silence, so no timeout value can ever be large enough.

```
# shot.py — direct reconnaissance
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url http://target.local/ --wait-until load --a11y

# rpa.py — propagated to shot.py, so scenarios reach the same targets
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario ./admin_login.json --wait-until load
```

A scenario can carry it as a root property instead, staying self-contained:

```
{"url": "http://target.local/", "wait_until": "load", "actions": [...]}
```

The CLI flag takes precedence over the scenario property.

Value	Waits for	Use when
`networkidle`	500 ms of network silence	default — keep it unless it fails
`load`	`load` event (page and sub-resources)	continuous polling / live statistics
`domcontentloaded`	HTML parsed, sub-resources still pending	very heavy page, DOM is all you need

Applies to the initial navigation only — the `naviguer` action is unaffected. `boussole.wait_until` reports the value only when it differs from the default.

---

8. Monitoring — structural checks

No image-based monitoring exists in Dinoer (no visual diff). Structural checks are text-based and CI-friendly.

8a. Continuous structural monitoring — `scripts/monitor-verifier.sh` (v1.18.0)

Monitors *structure* (`http_status`, `dom_stats`, `evaluations`) — zero image, zero LLM call, built on `--replay-verifier` (section 5h).

```
# First run — create the structural reference
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
```

```
  --scenario /opt/dinoer/scenarios/example_login.json \
  --sauver-verifier-reference /opt/dinoer/references/example_login.ref.json

# One check-and-alert pass — not a daemon, run it repeatedly via cron.
# On the git-clone channel, scripts/*.sh is never deployed to /opt/dinoer/,
# so it runs from the git source, as your own user. On the .deb channel,
# the three wrapped scripts (monter/demonter-repertoire-chiffre,
# monitor-verifier) ARE installed under /opt/dinoer/scripts/ — corrected
# 15/08/2026, this comment predates that channel's real construction.
bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/example_login.json \
  --reference /tmp/ref_sillage.json \
  --ntfy-topic dinoer-monitoring

# crontab -e (your own crontab)
*/15 * * * * bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/example_login.json \
  --reference /opt/dinoer/references/example_login.ref.json \
  --ntfy-topic dinoer-monitoring \
  >> /var/log/dinoer/cron-structural.jsonl 2>&1
```

Stable ⟶ silence. Regression ⟶ one `ntfy` notification with the diff. Each invocation is an isolated process — no daemon, no memory-leak risk, and Respectful Navigation caps reset cleanly on every pass.

Fixed 16/08/2026: the script used to call `rpa.py --no-capture --replay-verifier`, but `--no-capture` was no longer an `rpa.py` flag — every real invocation failed at argparse. The dead flag has been removed (Dinoer has no image path by default, so dropping it changes nothing else).

Guide-read lock nuance: if invoked under a distinct OS user (e.g. a system service account), that user needs `--guide-version` validated once (`~<home>/.config/dinoer/guide_state.json`).

---

9. Operation log

The log is `/var/log/dinoer/operations.jsonl`. If the configured journal path sits inside the encrypted directory and it is not mounted, entries are redirected to a local fallback (degraded write, 700/600) rather than written in clear text on the raw host.

```
# Read the last 10 entries
tail -n 10 /var/log/dinoer/operations.jsonl | python3 -m json.tool

# Filter by target (journal.py tool)
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com

# Filter mutating operations only
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --mutatif

# From a date
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --depuis 2026-07-01

# Failed runs only (v1.20.0) — result != success
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --erreurs
```

Fields in each entry:

Field	Meaning
`ts`	ISO 8601 timestamp
`operation_id`	Unique run identifier (v1.16.0), names the evidence directory `preuves/<AAAA-MM>/<id>/` when captures are archived
`outil`	`"shot.py"` or `"campagne.py"` only — `rpa.py` never journals itself, it runs through `shot.py`'s dispatch (corrected 15/08/2026 twice: first from a `mode` field the code never wrote, then to remove `"rpa.py"` as a possible value, itself never written)
`version`	Dinoer version
`cible_url`	Target URL
`resultat`	`"succes"` or `"echec"`
`mutatif`	`true` if at least one write action
`hostname_executant` / `utilisateur_executant` / `profil_actif`	Execution boussole (host, OS user, active operator profile)
`intention`	Label passed via `--intention` or scenario `intention` field — present only if set
`source_scenario`	Scenario file name only, no path (v1.18.0) — present only in RPA mode. Corrected 15/08/2026: previously the table also listed a `scenario` field (full path); the code never wrote one
`chainage`	List of `{scenario, profondeur, action_debut, action_fin}` — present only when `declencher_scenario` was used
`actions`	Summarised action list — present when the run had actions
`actions_raw`	Neutralised full action list — present only on a successful run with actions
`captures`	Reference(s) to structural captures — present only when the run produced any
`erreur`	Present only on failure
`respect`	The run's navigation ledger — present only when set
`evaluations`	Sanitised `{script, valeur_retournee}` values — present only if `evaluer` actions ran

No `duree_ms` field exists in the journal — corrected 15/08/2026, this table previously listed one the code never wrote. Per-action timing is `latences_actions` in the JSON output of a run, not in the journal entry.

9a. Log rotation (G-36)

Dinoer does not ship a logrotate configuration — `/var/log/dinoer/operations.jsonl` grows unbounded until the administrator installs one. `lib/journal.py` opens and closes the file on every write (no persistent file descriptor across runs), specifically so the **default** logrotate behaviour (rename the current file, create a fresh one) works correctly without any special option: the next write reopens the path and finds the new inode.

Do not add `copytruncate` to a Dinoer logrotate config — it is unnecessary here and reintroduces a write-loss window. Example `/etc/logrotate.d/dinoer`:

```
/var/log/dinoer/operations.jsonl {
    weekly
    rotate 8
    compress
    delaycompress
    missingok
    notifempty
    create 0640 dinoer dinoer
```

```
}
```

`journal.py` (the reader) already follows rotated files transparently (`operations.jsonl`, `.1`, `.2.gz`, …) — no extra step needed after rotation.

---

10. CLI flags — reference

shot.py

Flag	Default	Description
`--version`	—	Prints installed version and exits immediately — no Playwright, no other argument required (v1.18.0)
`--guide-version X.Y`	—	Proof of reading `docs/GUIDE_LLM.md` — required unless a valid local marker already exists (v1.18.0, section 1)
`--url URL`	required	URL to capture
`--actions FILE`	—	JSON file of sequential actions
`--action JSON`	—	Single action as inline JSON — quote carefully, prefer `--actions FILE` for JS-heavy actions
`--attendre-selecteur SEL`	—	Wait for a selector before finishing the run
`--timeout MS`	10000	Per-action Playwright timeout (ms)
`--wait-until VALUE`	`networkidle`	`networkidle`|`load`|`domcontentloaded` — initial navigation only (v1.22.0, section 7i)
`--largeur PX`	1280	Viewport width
`--hauteur PX`	720	Viewport height
`--a11y`	off	Include accessibility tree in JSON
`--stealth`	off	playwright-stealth stealth mode (v1.15.0)
`--secrets FILE`	—	Explicit path to a credentials file
`--auth-indicator SEL`	—	CSS selector present only in authenticated session
`--auth-indicator-negative SEL`	—	Requires `--auth-indicator`; CSS selector present only outside authenticated session
`--ignorer-waf`	off	A detected WAF block degrades `niveau_confiance` but no longer forces `pret_a_agir: false` on its own (v1.17.2, section 3e)
`--http-credentials`	off	Resolves HTTP Basic Auth credentials from the credentials file, scoped to the target's origin (v1.21.0, section 4g)
`--ignore-tls-errors`	off	Accept invalid TLS on controlled LAN/dev targets — never on public internet (v1.15.1)
`--no-evaluer`	off	Refuses the **evaluer** action (and `repli_js`) for the whole run — recommended against sensitive forms (v1.15.1)

Flag	Default	Description
`--no-filtre-evaluer`	off	Disables stdout neutralisation of **evaluer** return values, URLs and error messages — explicit debug runs only; when disabled, `boussole.filtre_evaluer_actif: false` is set (v1.23.0)
`--intention TEXT`	—	Business label recorded in the log
`--sauver-session FILE`	—	Saves cookies after actions
`--reprendre-session FILE`	—	Resumes a saved session
`--source-scenario NAME`	—	Internal (rpa.py plumbing for the journal — not for direct calls)
`--chainage JSON`	—	Internal (rpa.py plumbing for the journal — not for direct calls)

rpa.py

Propagates all relevant shot.py flags, plus:

Flag	Description
`--version`	Prints installed version and exits immediately (v1.18.0)
`--guide-version X.Y`	Proof of reading `docs/GUIDE_LLM.md` — checked independently, same rule as shot.py (v1.18.0)
`--scenario FILE`	Path to JSON or YAML scenario (required)
`--url URL`	Overrides scenario URL without modifying the file
`--stealth`	Propagated to shot.py
`--wait-until`	Propagated to shot.py (v1.22.0, section 7i)
`--ignorer-waf`	Propagated to shot.py (v1.17.2, section 3e)
`--http-credentials`	Propagated to shot.py; also settable as scenario root property `"http_credentials": true` (v1.21.0, section 4g)
`--auth-indicator-negative`	Requires an `auth_indicator` (CLI or scenario root property)
`--sauver-verifier-reference FILE`	Saves structural reference for `--replay-verifier` (v1.17.0, section 5h)
`--replay-verifier FILE`	Compares run against a structural reference, exit 1 on regression (v1.17.0, section 5h)
`--checkpoint FILE`	Resumes a long scenario after a mid-run failure (v1.17.0, section 5i)

campagne.py (research pipeline)

Flag	Description
`--manifeste FILE`	Campaign manifest (JSON) — requires `id_campagne` + `cibles`
`--id-campagne ID`	Campaign identifier (used in the manifest and extraction)
`--extraire-cible DEMANDE`	Targeted extraction on an already-collected corpus, without synthesis. Requires `--id-campagne` or `--corpus`
`--corpus FILE`	Direct path to a `collecte.jsonl` — alternative to `--id-campagne` for `--extraire-cible`
`--format-extraction {json,markdown,html}`	Output format for `--extraire-cible` (default: `json`) — added 15/08/2026
`--desactiver-cache`	Bypass the search cache

Flag	Description
`--purger-cache`	Purge the whole search cache
`--purger-cache-avant-jours N`	Purge cache entries older than N days

Target types in the manifest: `query`, `url`, `produit`, `table_reference`. Artefacts: the shared `/var/log/dinoer/operations.jsonl` + a per-campaign `collecte.jsonl`. Full detail: `campagne.py --help`.

---

11. Exit codes and output

Exit codes

Code	Cause	What to do
0	Success	—
1	Playwright error, failed action, rpa.py assertion, `action_secret_en_clair`	Read `erreur` in JSON. See `GUIDE_LLM_INTERACTIONS.md`
1	`guide_non_lu` — missing/wrong `--guide-version`, no valid marker (v1.18.0)	Fires before Playwright launches. Read `docs/GUIDE_LLM.md`, relaunch with `--guide-version X.Y` (section 1)
2	Incompatible arguments, `arguments_incompatibles`, `url_scheme_interdit`, `chemin_sensible_refuse`	Read `message` — it names the conflict
3	`playwright` module not found	Invoke via `/opt/dinoer/venv/bin/python`
42	`SecretsFermesError` — encrypted directory not mounted, or invalid checksum	Mount it, or verify the credentials file
43	`SecretsNonConfigureError` — no `secrets_dir` configured	Configure `secrets_dir` in `dinoer.conf` (undo a missing sample: create `/opt/dinoer/dinoer.conf`)

Output JSON structure

```
{
  "succes": true,
  "http_status": 200,
  "url_finale": "https://target.local/dashboard",
  "erreurs_js": [],
  "erreurs_console": [],
  "duree_ms": 2400,
  "horodatage": "2026-07-01T12:00:00+02:00",
  "dom_stats": {"boutons": 14, "inputs": 9, "listes_deroulantes": 2, "formulaires": 1,
  ↪  "liens": 41, "dialogues": 0},
  "a11y_tree": "...",
  "evaluations": [],
  "extraction_texte": null,
  "latences_actions": [
    {"index": 0, "type": "naviguer", "latence_ms": 842},
    {"index": 1, "type": "cliquer", "latence_ms": 63}
  ],
```

```
  "respect": {
    "pages_visitees": 0,
    "actions_executees": 3,
    "duree_totale_ms": 2400,
    "indice_agressivite": 0.33
  },
  "etat": {
    "pret_a_agir": true,
    "niveau_confiance": "eleve",
    "raisons": ["aucun signal de friction détecté"]
  },
  "boussole": {
    "utilisateur": "operator",
    "ip_locale": "__IP_LAN__",
    "repertoire": "/opt/dinoer",
    "operation_id": "a1b2c3d4e5f6",
    "url_courante": "https://target.local/dashboard",
    "titre_page": "Dashboard — My App",
    "dernier_code_http": 200,
    "stealth_actif": true,
    "auth_status": "active",
    "respect": { "pages_visitees": 0, "actions_executees": 3, "duree_totale_ms": 2400,
    ↪  "indice_agressivite": 0.33 }
  },
  "dinoer_meta": {
    "version_shot": "1.0.1",
    "horodatage_iso": "2026-08-12T14:23:11+02:00",
    "hostname_executant": "operator-host",
    "utilisateur_executant": "operator",
    "profil_actif": "operateur.exemple.yaml",
    "url_au_moment_capture": "https://target.local/dashboard"
  }
}
```


`operation_id` (v1.16.0) is always present and identifies this run uniquely — it matches the `operation_id` field of this run's entry in the operations log (section 9), and names the evidence directory `preuves/<AAAA-MM>/<operation_id>/` when captures are archived (corrected 15/08/2026, verified against `lib/journal.py`: no `/tmp/dinoer/<operation_id>/` directory exists anywhere in the code — `/tmp/dinoer/` only ever holds the operations-log fallback file). `etat` (v1.16.0) is present on the success path only. `latences_actions` (v1.20.0) is always present (empty list if no actions), one entry per action that actually dispatched — see `GUIDE_LLM_MONITORING.md` for how it complements `respect.duree_totale_ms`.

Conditional keys (absent when inactive): `dom_stats`, `a11y_tree`, `evaluations`, `extraction_texte`, `auth_status`, `stealth_actif`, `session_derive`, `respect.plafond_atteint`, `respect.waf_bloquants`, `respect.indice_agressivite` (present whenever at least one action ran), `boussole.repli_js_utilise`, `boussole.wait_until`, `boussole.http_credentials_actif`, `boussole.http_auth_requise`, `boussole.tls_errors_ignored`, `boussole.waf_ignore_actif`, `boussole.filtre_evaluer_actif`, `boussole.champs_rediges`, `actions_executees_avant_echec`, `pages_visitees_avant_echec` (failure JSON only, v1.17.0). See `GUIDE_LLM_MONITORING.md` for the exhaustive activation table.

Error — format


```
{
  "succes": false,
  "erreur": "secrets_fermes",
  "message": "Le répertoire chiffré Dinoer est initialisé mais non monté.",
  "code_sortie_recommande": 42,
```

```
  "boussole": { "url_courante": "", "titre_page": "" }
}
```

Reference paths

Path	Role
`/opt/dinoer/`	Production installation
`/opt/dinoer/venv/bin/python`	Python to use for every invocation
`/opt/dinoer/dinoer.conf`	Machine configuration (secrets_dir, navigation, ntfy)
`/opt/dinoer/scenarios/`	RPA scenarios (including `diagnostic_dom.json`)
`/opt/dinoer/docs/`	Documentation
`/opt/dinoer/references/`	`--sauver-verifier-reference` / replay references
`/tmp/dinoer/`	Session files (`--sauver-session`/`--reprendre-session`) and the operations-log fallback file — corrected 15/08/2026: no per-`operation_id` subdirectory exists here, see the next row
`/var/log/dinoer/preuves/<AAAA-MM>/<operation_id>/`	Evidence directory for one run, isolated by `operation_id` (v1.16.0), created only when captures are archived
`~/Vaults/__PROJET__/Dinoer/`	Credentials + log (gocryptfs volume)
`~/git/Dinoer/Dinoer/`	Git sources (edit here, then `deploy.sh`)
`/var/log/dinoer/operations.jsonl`	Persistent operation log (`journal.py`)

Deploy after modifying sources:

```
bash ~/git/Dinoer/Dinoer/scripts/deploy.sh
```