Dinoer — Dokumentation

Souveräne, lokale Web-Recherche für LLM-Agenten

Version 1.0.1

2026-09-25

Übersetzung. Maßgeblich ist die englische Fassung.

Über diese Zusammenstellung

Dieses Dokument fasst vier Texte zusammen, die auch einzeln bestehen und sich je an eine andere Leserschaft richten. Die *Kurzreferenz* beginnt mit den Befehlen selbst, für alle, die sofort einen brauchen. Die *Übersicht* stellt Dinoer vor und installiert es. Das *Bedienerhandbuch* erklärt, was Sie delegieren und warum sich das Werkzeug so verhält. Das *Betriebshandbuch* ist die Referenz: Befehle, Aktionen, Optionen, Rückgabecodes. Jedes beginnt mit einer eigenen Einleitung, weil jedes auch allein gelesen wird.

Inhaltsverzeichnis

Dinoer — Spickzettel

Version 1.0.1 — September 2026

Alles auf einer Seite. Vollständige Referenz: `docs/MANUEL.md`.

Drei Befehle

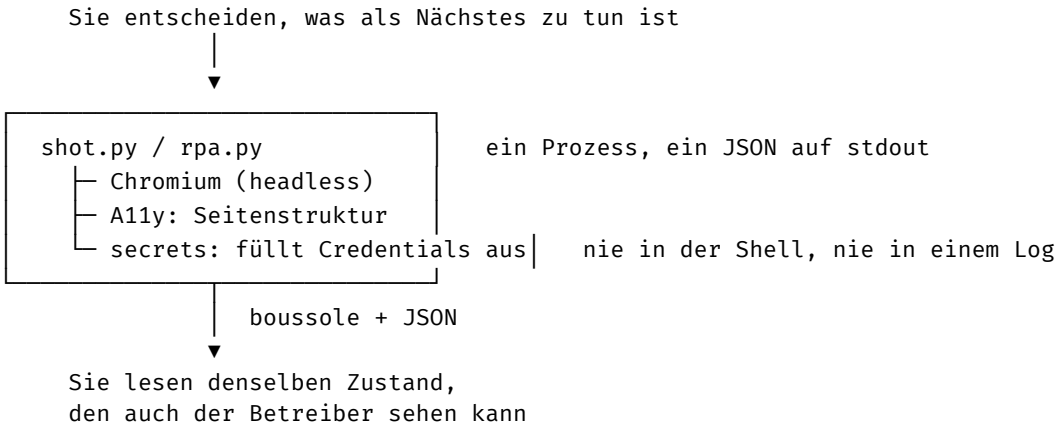
```
# Eine Seite sehen: Accessibility-Baum
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url URL --a11y

# Ein Szenario ausführen
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py --scenario DATEI.json

# Eine Szenario-Referenz lesen und vergleichen (strukturelle Überwachung)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario DATEI.json --replay-verifier REF.json
```

Der erste Aufruf auf einer Maschine benötigt `--guide-version X.Y`, zu lesen mit `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`.

Die Schleife



Der Sitzungszustand lebt in einer Datei, nicht im Prozess: ein zweiter Aufruf mit `--reprendre-session` nutzt Cookies wieder — nie den DOM-Zustand.

Die Ausgabe in dieser Reihenfolge lesen

Lesen	Sagt Ihnen
`succes`	ob der Lauf abgeschlossen wurde
`boussole.url_courante`	wo Sie tatsächlich gelandet sind
`boussole.dernier_code_http`	Status der letzten Navigation
`etat.pret_a_agir` + `etat.raisons`	wahrgenommene Friktionen — ein Bericht, nie eine Schranke
`a11y_tree`	Seitenstruktur — Überschriften, Felder, Schaltflächen
`respect`	Ihr eigener Fußabdruck: Seiten, Aktionen, Dauer

Stimmt `boussole` nicht mit Ihrer Erwartung überein: vor jeder mutierenden Aktion anhalten.

Jede Aktion

`type` ist immer erforderlich. Die folgenden Schlüssel sind die zusätzlichen.

Aktion	Erforderlich	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 — die einzig korrekte Form

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

Nie ein Secret in die Shell extrahieren. `lib/repertoire_chiffre.py` löst es innerhalb des Playwright-Prozesses auf; der Wert erreicht nie Ihre Kommandozeile, Ihre History oder ein Log.

Wenn sich etwas sperrt

Symptom	Versuchen
Klick läuft in Timeout, Element visuell verborgen	`"force": true`, dann `"repli_js": true`
Element unterhalb des sichtbaren Bereichs	zuerst `defiler`
Seite wird nie fertig geladen	`--wait-until load`
Absenden bewirkt nichts, kein Fehler	native HTML-Validierung — Formular über `evaluer` absenden
`exit 42`	verschlüsseltes Verzeichnis nicht gemountet (`bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`), oder Prüfsumme der Credentials ungültig (Credentials-Datei prüfen) — beides `SecretsFermesError`
`guide_non_lu`	einmalig `--guide-version` übergeben
403 / 429	`respect.waf_bloquants` lesen — ein Signal, keine Ausnahme

Exit-Codes

`0` Erfolg · `1` Ausführung fehlgeschlagen oder Assertion fehlgeschlagen · `2` Ungültige Argumente (abgelehnt, bevor ein Browser gestartet wurde) · `3` Falscher Interpreter – verwenden Sie die venv (`/opt/dinoer/venv/bin/python`) · `42` Verschlüsselter Credential-Verzeichnis geschlossen oder Prüfsumme der Credentials ungültig (`SecretsFermesError` Familie) · `43` Keine `secrets_dir` konfiguriert (`SecretsNonConfigureError`).

Dinoer — souveräne, lokale Web-Recherche für LLM-Agenten

> **Für den menschlichen Betreiber**: Dinoer läuft auf Ihrer eigenen Maschine, delegiert Suche und Sammlung an Primitiven, die Sie Zeile für Zeile lesen können, und liefert Ihnen einen belegten, datierten Markdown-Bericht — keine Blackbox-Antwort.
>
> **Für das LLM**: `docs/GUIDE_LLM.md` ist Ihre operative Referenz. Beginnen Sie dort.

Was ist Dinoer?

Dinoer ist eine **passive**, **lokale**, **souveräne Such- und Synthese-Engine**. Es ist ein Fork von Diwall (visuelle Browser-Automatisierung für LLMs), dem die gesamte Wahrnehmungsschicht entzogen wurde — **null Screenshots**, **null Set-of-Mark**, **null Vision-Modell**. Dinoer betrachtet eine Seite nie; es liest sie: DOM, Accessibility-Baum und bereinigter Seitentext.

Wo Diwall die Frage beantwortet „mit einer authentifizierten Oberfläche visuell interagieren", beantwortet Dinoer eine andere Frage: „eine große Zahl öffentlicher Quellen erkunden und daraus ein belegtes, überprüfbares Signal zusammenstellen" — auf Hardware so bescheiden wie einem Raspberry Pi 5.

```
Anfrage → SearXNG-Entdeckung → leichte HTTP-Sammlung
        → Eskalation zu einem echten Browser nur für Seiten, die ihn brauchen
        → Synthese durch ein delegiertes LLM → datierter, belegter Markdown-Bericht
```

Doktrin: Der Python-Code trägt keine Geschäftslogik. Jedes Modul erledigt eine mechanische Aufgabe — SearXNG abfragen, bereinigten Text aus einer Seite extrahieren, ein verschlüsseltes Credential lesen, eine Benachrichtigung senden. Die *Strategie* einer Recherche (wie nachgefasst wird, wann eskaliert wird, wann gestoppt wird) lebt in einem Szenario, niemals fest im Modulcode. Siehe `docs/GUIDE_LLM.md` für die vollständige Doktrin.

Positionierung: Worauf Dinoer setzt und worauf nicht.

Dinoer konkurriert nicht mit generellen Suchassistenten (wie Perplexity und ähnlichen) hinsichtlich der Breite, des Umfangs oder des Preises bei der Informationsbeschaffung. Ein echter Test (14. August 2026, Reputationsforschung zu einem realen Thema) hat dies direkt gemessen, anstatt es nur anzunehmen: Von den 28 Seiten, die durch Dinoers eigene, von SearXNG gesteuerte Suchfunktion gefunden wurden, waren drei Quellen, die eine einfache, nicht vorbereitete Perplexity-Abfrage sofort lieferte (ein LinkedIn-Profil, eine Projektseite, ein Stockfoto-Hinweis), vollständig abwesend – dies war auf SearXNG-Abfragen zurückzuführen, die nach der falschen Art von Suchergebnissen suchten (Unternehmensverzeichnisse, nicht den Begriffen, die diese Seiten hätten liefern können), und nicht auf einen Ranking- oder Kürzungsmangel im weiteren Verlauf. Ein generischer Such-Backend mit authentifizierten, cookie-basierten Engines dahinter hat eine strukturelle Reichweite, die eine nicht authentifizierte lokale SearXNG-Instanz nicht besitzt.

Was derselbe Test auf demselben Korpus verifiziert hat, misst eher als dass er etwas annimmt: **eine nachvollziehbare, reproduzierbare Synthese eines fixierten Korpus**. Jede Aussage in einem Dinoer-Bericht ist einer bestimmten Seite zuzuordnen, die tatsächlich auf die Festplatte geschrieben wurde (`collecte.jsonl`/`operations.jsonl`) – ohne jegliche Abhängigkeit von dem, was ein Such-Backend eines Drittanbieters während der Generierung der Antwort getan hat. Eine direkte Überprüfung des gesamten Ereignisstroms des delegierten Modells während der Synthese (nicht nur seines endgültigen Textes) bestätigte, dass während der Berichtserstellung keine externen `websearch`/`webfetch` Aufrufe den Korpus erreicht haben. Das ist das eigentliche Wertversprechen: genau zu wissen, woher eine Antwort stammt, und nicht mehr als ein allgemeines Tool liefern würde.

Architektur

```
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` behalten den ReAct-Ausführungskern von Diwall (`naviguer`, `remplir`, `cliquer`, `evaluer`, Sitzungspersistenz, Credential-Auflösung) — nichts von dessen Wahrnehmungsschicht.

Fähigkeiten

Funktion	Beschreibung
SearXNG-Entdeckung	Reine HTTP-Abfrage gegen eine lokale oder entfernte SearXNG-Instanz — kein Browser-Overhead für die Suche
Leichte Sammelstufe	`requests` + BeautifulSoup-Extraktion, `robots.txt`-bewusst, WAF-bewusst

Funktion	Beschreibung
Eskalation zur schweren Stufe	Playwright, nur für Seiten eingesetzt, die die leichte Stufe nicht lesen konnte (JS-gerenderte Shells)
Semantische Textextraktion	Aktion `extraire_texte` — bereinigter Haupttext, kein Screenshot
Accessibility-Snapshot	`--a11y` — semantische Seitenstruktur (A11y-Baum), es entsteht nie ein Bild
Gezielte Extraktion	`lib/extraction.py` — strenger `trouve/valeur/url`-Vertrag, erklärt Abwesenheit statt eine Antwort zu erfinden
Referenzseiten-Tabellen	`lib/tables_reference.py` — eine persistente, belegte Tabelle bekannter Seiten pro Thema
Vektor-Suchcache	`lib/cache_recherche.py` — ChromaDB-gestützt, vermeidet erneute Abfragen für nahezu identische Anfragen
Deduplizierung & Aktualität	Deduplizierung auf Kampagnenebene nach exakter URL, Obergrenze pro Hostname, 30-Tage-Aktualitätsfenster vor erneutem Crawl
Respektvolles Crawling	Zufällige Verzögerung zwischen Zielen, harte Verweigerung bei WAF/robots.txt-Signalen — nie umgangen
Credential-Auflösung	Sichere Credential-Injektion — nie im Klartext, nie auf der Kommandozeile
Verschlüsseltes Verzeichnis	gocryptfs-Volume — `SecretsFermesError` (Exit 42), wenn es nicht gemountet ist
Vorgangsprotokoll	Persistentes Append-only-Protokoll aller Läufe — wer hat was, wo, wann getan
RPA-Szenarien	Aktionssequenzen aus JSON-Dateien ausführen, für den Eskalationspfad zur schweren Stufe
Cross-Origin-iframes	`cliquer_iframe` / `remplir_iframe` zielen auf Elemente innerhalb von iframes
TOTP / asynchrones MFA	Credential-geschützte Ziele bleiben erreichbar, wenn ein Lauf der schweren Stufe sich authentifizieren muss

---

Berichtsqualität: Automatisch generierter Entwurf vs. sorgfältige Recherche

`campagne.py`'s eigener Abschlussbericht (`lib/synthese.py::construire_contexte()` erstellt und kürzt den Korpus, `rediger_rapport()` verfasst anschließend den Text) ist ein **Arbeitsentwurf**, nicht das fertige Ergebnis: er verkettet den gesammelten Korpus in Dateireihenfolge, abgeschnitten bei 4000 Zeichen/Seite und 60.000 insgesamt – ohne Relevanzbewertung. Bei einem großen, verrauschten Korpus führt dies zuverlässig dazu, dass generische oder themenfremde Seiten vor den eigentlichen Quellen angezeigt werden, und kann stillschweigend die relevantesten Einträge hinter dem Abschneidepunkt auslassen.

Bei einer realen Forschungaufgabe (eine lokale Veranstaltungsliste, siehe ”Positionierung” oben für eine Aufgabe, bei der das Ergebnis anders ausfiel), übertraf die Qualität des Berichts nachweislich ein allgemeines Suchwerkzeug (Perplexity) – aber dieser Bericht wurde **nicht** durch einen einzigen `campagne.py` Durchlauf erstellt. Er stammt von einem Operator, der `campagne.py --extraire-cible` verwendete – Dutzenden einzelner, offener Abfragen an denselben Datensatz, wobei jedes Mal das delegierte Modell selbst beurteilen konnte, ob es eine einmalige Tatsache oder ein mehrtägiges Ereignis las – gefolgt von einer manuellen Zusammenführung der Ergebnisse. Siehe `docs/GUIDE_LLM.md` für das genaue Extraktionsmuster.

Wenn Sie eine schnelle, nicht kritische Zusammenfassung benötigen, ist der automatische Bericht ein guter Ausgangspunkt. Wenn Sie einen Bericht benötigen, dem Sie ohne Aufsicht vertrauen können, verwenden Sie stattdessen das gezielte, wiederholte Extraktionsmuster.

Voraussetzungen

Komponente	Version / Anmerkungen
Betriebssystem	Debian 13 Trixie (Linux)
Python	3.11+ in isoliertem venv (PEP 668 — System-pip unter Debian 13 gesperrt)
Playwright	1.62+ (im venv installiert) — nur vom Eskalationspfad der schweren Stufe genutzt
Chromium	Headless, installiert über `playwright install chromium`
SearXNG	Eine erreichbare Instanz (lokal oder entfernt), HTTP-JSON-API
Ollama	Lokales, CPU-freundliches Embedding-Modell (`nomic-embed-text`) für den Suchcache — kein Vision-Modell, keine GPU erforderlich
OpenCode	Delegiertes Reasoning-Backend für die Berichtssynthese (standardmäßig kostenlose Modelle)

Keine GPU erforderlich. Das Referenzziel ist ein Raspberry Pi 5 mit 8 GB RAM.

Installation

Zwei Kanäle, die sich gegenseitig ausschließen, wenn sie auf derselben Maschine verwendet werden.

`.deb` package — der normale Pfad, wenn Sie Dinoer unverändert verwenden möchten:

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

Installiert den Systembenutzer und die Gruppe `dinoer`, eine isolierte Python-virtuelle Umgebung, Chromium, die sechs `dinoer-*` Befehle und deren Handbücher in vier Sprachen. Pakete, Quellcode und Prüfsummen werden auf dinoer.davalan.fr veröffentlicht – siehe die Seite Downloads für Details, einschließlich der Bedeutung des `apt` Sandbox-Hinweises.

Git clone – wenn Sie den Code ändern möchten:

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

Dies erstellt den Systembenutzer und die Systemgruppe `dinoer`, die virtuelle Umgebung, stellt den Code unter `/opt/dinoer/` bereit und führt einen Funktionstest aus (`shot.py --a11y` gegen eine echte URL).

Die Konfiguration befindet sich in `/etc/dinoer/dinoer.conf` (Kanal `[.deb]`) oder `/opt/dinoer/dinoer.conf` (git-clone-Kanal); eine Beispielkonfiguration wird daneben installiert, nämlich als `dinoer-sample.conf` – reines JSON, nicht kommentiert (korrigiert am 15./08/2026]: JSON hat keine Kommentarsyntax, die Datei war nie kommentiert). Ausnahme: `campagne.py` liest niemals `DINOER_CONF` oder den oben genannten git-clone-Pfad – es liest `/opt/dinoer/dinoer.conf` fest codierte Werte und löst seine eigenen Pfade über spezielle Umgebungsvariablen auf (`DINOER_CAMPAGNES_DIR`, `DINOER_SEARXNG_URL`, `DINOER_TABLES_REFERENCE`, `DINOER_JOURNAL`).

Deinstallation

```
bash scripts/uninstall.sh --dry-run   # Vorschau, keine Änderungen
bash scripts/uninstall.sh             # interaktive Bestätigung
```

Entfernt: `/opt/dinoer/`, `/var/log/dinoer/`, Systembenutzer `dinoer`, Systemgruppe `dinoer`. **Nie angerührt**: `~/Vaults/` (Ihre Credentials), das Repository selbst.

Benutzung (durch Ihr LLM)

Semantische Extraktion, kein Bild

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

Eine Recherchekampagne

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

Vollständige LLM-Referenz: `docs/GUIDE_LLM.md`

Credentials

Credentials werden in JSON-Dateien gespeichert, eine pro Domain, **nie im Code oder in Szenariodateien**:

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

In einem Szenario oder einer Aktion: `"valeur": "depuis_secrets", "secret_cle": "password"` — Dinoer liest das Credential zur Laufzeit aus dem Credentials-Verzeichnis.

Der Pfad ist über `/opt/dinoer/dinoer.conf` oder die Umgebungsvariable `DINOER_SECRETS_DIR` konfigurierbar.

Empfehlung: Schützen Sie `~/Vaults/__PROJET__/Dinoer/` mit `chmod 700` und verschlüsseln Sie es mit `gocryptfs` (siehe `scripts/configurer-repertoire-chiffre.sh --gocryptfs` — nur Git-Clone-Kanal, nicht vom `.deb` mitgeliefert; richten Sie auf diesem Kanal `gocryptfs` selbst ein und zeigen Sie `secrets_dir` auf den gemounteten Pfad). Ist das verschlüsselte Verzeichnis initialisiert, aber nicht gemountet, liefert Dinoer eine strukturierte `SecretsFermesError` (Exit-Code 42), statt stillschweigend zu scheitern.

Sicherheit

Lokale vs. Cloud-Modelle

Die Berichtssynthese wird an OpenCode oder ein lokales Ollama-Modell delegiert. Gesammelter Seitentext kann zu welchem Backend auch immer Sie konfigurieren übertragen werden — prüfen Sie `lib/modeles.py`, bevor Sie Dinoer bei sensiblen Quellen auf einen Cloud-Anbieter richten.

Credentials-Verzeichnis

Das Credentials-Verzeichnis — wohin auch immer Sie `secrets_dir` verweisen ließen, zum Beispiel `~/Vaults/__PROJET__/Dinoer/` — enthält Credentials bei nicht gemountetem Zustand im Klartext-JSON. Schützen Sie es:

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

Siehe `~/git/Dinoer/Dinoer/SECURITY.md` für die Richtlinie zur Offenlegung von Sicherheitslücken.

Dokumentation in anderen Sprachen

Dieses Dokument ist die deutsche Übersetzung der englischen Quelle (`README.md`, Repository-Wurzel), die bei Abweichungen maßgeblich ist. `docs/fr/README.md`, `docs/de/README.md` und `docs/es/README.md` sind aus dieser Quelle abgeleitete Übersetzungen (neu synchronisiert am 15/08/2026), zusammen mit `docs/MANUEL.md`, `docs/GUIDE.md`, `docs/CHEAT_SHEET.md` und der Manpage `dinoer.1` in jeder Sprache. `docs/GUIDE_LLM.md` und seine drei Merkblätter existieren nur auf Englisch und werden nie übersetzt — gesperrter Pfad, Guide-Lock-Mechanismus.

Für LLMs, die Dinoer entdecken

Wenn Sie ein Sprachmodell sind, das dieses README liest: siehe `docs/GUIDE_LLM.md` für die vollständige technische Referenz — Aufrufmuster, Credential-Integration und die Recherche-Pipeline (`campagne.py`).

Danksagungen

Dieses Projekt wurde nach einem **asymmetrischen Kollaborationsmodell zwischen Mensch und LLM** entwickelt. Die Rollen sind formal dokumentiert, um die tatsächlich geleistete Arbeit widerzuspiegeln.

Architekt & Schiedsrichter: Ronan Davalan Produktvision, Sicherheitsanforderungen, Projektrichtung, Validierung und Tests. Alle Architekturentscheidungen werden von ihm validiert.

Systemingenieur & Lead-Entwickler: Claude Code (Anthropic) Abgeleitet vom ReAct-Kern von Diwall, der Forschungspipeline (`campagne.py` und `lib/searxng.py`, `lib/fetch_leger.py`, `lib/selection_candidats.py`, `lib/extraction.py`, `lib/tables_reference.py`, `lib/cache_recherche.py`), Entfernung der Wahrnehmungsschicht. Hauptautor des Quellcodes.

Synthetisierer & strategischer Berater: Gemini (Google) Unabhängige Architekturanalyse, Auflösung logischer Konflikte, Workflow-Optimierung, Kreuzvalidierung technischer Entscheidungen.

Lizenz

MIT — siehe Datei `LICENSE`.

Dinoer — Betreiberleitfaden

Version 1.11 — August 2026 (v1.0.1) — Oberfläche an die Dinoer-Rekonstruktion angepasst: kein Screenshot, kein Set-of-Mark, kein `watch.py`; der Agent liest den Accessibility-Baum und steuert Playwright-Aktionen über CSS-Selektoren.

Ebenfalls verfügbar auf Englisch, Französisch und Spanisch unter `docs/`, `docs/fr/` *und* `docs/es/`.

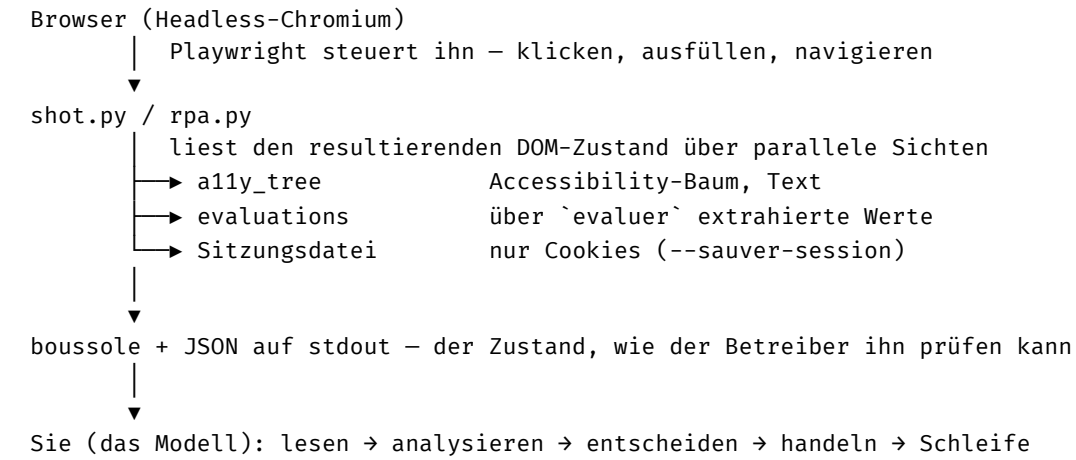
Warum Dinoer — was Sie tatsächlich delegieren

Das Problem, das Dinoer löst

Wenn Sie mit einem LLM an einer Webanwendung arbeiten, entsteht eine Wahrnehmungsasymmetrie: Das Modell liest Code, führt Befehle aus, beobachtet Textausgaben — aber es sieht nicht die Oberfläche, die Ihre Nutzer sehen. Sie schon.

Diese Asymmetrie erzeugt eine spezifische Form von Unsicherheit: Sie wissen nicht, ob das, was das Modell beschreibt, dem entspricht, was Sie in einem Browser sehen würden. Um sicherzugehen, müssen Sie ihm entweder aufs Wort glauben oder es selbst überprüfen.

Dinoer löst dieses Problem, indem es dem Modell dieselbe strukturierte Sicht verschafft, die Sie in einem Browser hätten: den Accessibility-Baum, gelesen über ein echtes Headless-Chromium, plus die DOM-Werte, die es mit `evaluer` extrahiert. Sie nehmen das Modell nicht mehr beim Wort — Sie beobachten denselben Zustand wie es.



Was Sie delegieren

Dinoer lässt Sie **repetitive und kontrollintensive Überprüfungen** delegieren:

- Prüfen, dass 20 Seiten einer Site nach einem Deployment korrekt antworten
- Bestätigen, dass ein Login-Formular auf der richtigen Oberfläche funktioniert
- Sicherstellen, dass ein Deployment die Struktur einer kritischen Ansicht nicht beschädigt hat
- Ein Admin-Panel über dieselbe Oberfläche steuern, die ein Mensch nutzen würde

Ohne Dinoer liegen diese Überprüfungen in Ihrer Verantwortung. Mit Dinoer führt das Modell sie aus und meldet das Ergebnis — mit dem JSON-Beleg dazu.

Was bei Ihnen bleibt

Bei Ihnen bleibt die **Sinnvalidierung auf hoher Ebene**: die Entscheidung, ob das vom Modell präsentierte Ergebnis akzeptabel ist, Ihren Erwartungen entspricht und mit dem übereinstimmt, was Ihre Nutzer sehen sollten. Diese Entscheidung bleibt Ihre.

Respektvolle Navigation (v1.15.0)

Dinoer verschleiert seine Identität nicht, um Bot-Erkennung zu umgehen. `--stealth` entfernt automatische technische Markierungen (`navigator.webdriver`), die Headless-Browser unabhängig von der Absicht blockieren — es ändert weder die IP-Adresse noch die Identität des Betreibers noch die Tatsache, dass der Lauf deklariert ist. Im Gegenzug meldet jeder Lauf seinen eigenen Fußabdruck (`respect`: besuchte Seiten, ausgeführte Aktionen, Dauer) und respektiert konfigurierbare Höflichkeitsverzögerungen und harte Obergrenzen (`dinoer.conf [navigation]`). Das Recht zu navigieren und die Pflicht, dies messbar zu tun, werden als untrennbar behandelt — siehe `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 für den Praxiskontext, der dies geprägt hat.

Lokale Ziele — die Höflichkeitsverzögerung ist keine Doktrin, sondern ein Standardwert (v1.19.0): Der mitgelieferte Wert `min_action_delay_ms: 800` schützt einen unkonfigurierten Erstlauf gegen das öffentliche Internet — gegen Ihre eigene Entwicklungs- oder Produktionsmaschine ist er bedeutungslos. Setzen Sie ihn für lokales Debugging in Ihrer lokalen `dinoer.conf` auf `0`; siehe `docs/MANUEL.md` Abschnitt 3b.

Wann Dinoer das richtige Werkzeug ist

Anwendungsfall	Für Dinoer geeignet?
Strukturelle Validierung nach einem Deployment	✅ Ja
Diagnose einer defekten Interaktion	✅ Ja
Navigation und Formulareingabe (~30 s max.)	✅ Ja
Delegation repetitiver Prüfungen	✅ Ja
Lange Server-Operation (Klonen ~2–5 min)	❌ Nein — Playwright-Timeout
Massenlöschung oder -mutation	❌ Nein — direkten API-Aufruf bevorzugen
Workflow, der ein Rollback erfordert	❌ Nein — Dinoer kann nichts rückgängig machen

---

Dieses Dokument ist für die Person geschrieben, die Dinoer betreibt.

Es ergänzt `GUIDE_LLM.md` (für Modelle bestimmt) um konkrete Beispiele, Schritt-für-Schritt-Anleitungen und Hinweise zu häufigen Stolperfallen.

Demonstrations-Anwendungsfälle

Die folgenden Fälle veranschaulichen, wie eine Sitzung aus Agent plus Dinoer in der Praxis aussehen kann. Sie sind dazu gedacht, dass Sie sie gegen Ihren eigenen Kontext bewerten — nicht als Empfehlung, einen bestimmten Fall zu übernehmen. Nur Fall 1 wird als lauffähiges Szenario ausgeliefert; die anderen sind bewusst erzählend, und jeder erklärt unter seiner eigenen Überschrift, warum.

Fall 1 — lokale CSS/JS-Fehlersuche

Als reales, lauffähiges Szenario eingecheckt: `scenarios/exemples/depannage_local.json`. Es diagnostiziert eine Layoutverschiebung oder eine blockierte Interaktion auf einer lokal bereitgestellten Oberfläche — eine schnelle Sonde, die `erreurs_js`/ `erreurs_console` und den Accessibility-Baum liest und dann die Korrektur mit `rpa.py --replay-verifier` gegen eine vor der Regression erfasste Referenz validiert. Direkt ausführen:

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

Fall 2 — Hardwarekomponenten über Shops hinweg vergleichen

Ein Agent, der gebeten wird, Preis und Lagerbestand einer Komponente über mehrere Online-Shops hinweg zu vergleichen, könnte Dinoer mit einem separaten Werkzeug zur URL-Entdeckung (zum Beispiel einer lokalen Suchinstanz) kombinieren, um Kandidatenseiten von Shops zu finden, dann Dinoer im schreibgeschützten Modus mit `evaluer`-Aktionen nutzen, um Preis, Lagerbestand und Spezifikationen von jeder Seite zu extrahieren, und schließlich die Ergebnisse selbst vergleichen.

Bewusst nicht als eingechecktes Szenario ausgeliefert: Einen bestimmten Shop in einem öffentlichen, versionierten Szenario zu nennen, ist eine Entscheidung, die Ihnen gehört, kein Standard, den dieses Projekt in Ihrem Namen treffen sollte. Es birgt auch ein reales Fragilitätsrisiko — ein öffentliches Szenario, das auf eine namentlich genannte kommerzielle Site zielt, kann Monate später scheitern, wenn sich deren Anti-Bot-Haltung ändert (39 % der in `docs/RETOUR_EXPERIENCE.md` FR-77 erfassten kommerziellen Sites gaben einen sofortigen Block zurück), was das Beispiel eher diskreditiert als hilft. Wenn Sie diese Komposition selbst aufbauen: Beachten Sie, dass jedes Werkzeug zur URL-Entdeckung, mit dem Sie Dinoer kombinieren (eine lokale Suchinstanz oder sonstiges), keine Dinoer-Komponente ist — es ist ein separates Teil, das der Agent obendrauf komponiert.

Fall 3 — technische Dokumentation erkunden und zusammenfassen (Single-Page-Apps)

Ein Agent, der beauftragt ist, einen Integrationsleitfaden für eine als Single-Page-App gebaute Dokumentations-Site zu erstellen, könnte `rpa.py` mit `attendre_reseau_calme` nutzen, um clientseitiges Routing sich setzen zu lassen, den Accessibility-Baum extrahieren, um die Seitenstruktur zu kartieren, dann Codeblöcke rekursiv mit `evaluer` durchlaufen, um ihren exakten Inhalt zu ziehen, und schließlich das gesammelte Material zu einem Leitfaden synthetisieren.

Aus demselben Grund wie Fall 2 nicht als eingechecktes Szenario ausgeliefert — eine bestimmte Dokumentations-Site zu nennen (oder, schlimmer, einen bestimmten Zahlungsanbieter, dessen Dokumentation zufällig das funktionierende Beispiel ist) ist eine kommerzielle und reputationsbezogene Festlegung, die dieses Projekt nicht standardmäßig treffen sollte, und dasselbe WAF-Fragilitätsrisiko gilt für ein öffentliches Szenario, das auf ein einziges reales Ziel festgenagelt ist.

Fall 4 — ein selbstgehostetes Observability- oder Analytics-Dashboard konfigurieren

Ein Betreiber, der ein selbstgehostetes Monitoring- oder Web-Analytics-Dashboard hinter einem Reverse-Proxy einrichtet, kann Dinoer nutzen, um die Oberfläche selbst zu steuern — ein Dashboard erstellen, eine Datenquelle verdrahten, eine Alarmregel setzen — genauso, wie jedes andere Admin-Panel konfiguriert wird, statt Dateien für Schritte von Hand zu bearbeiten, die die Oberfläche eigentlich übernehmen soll. Dies schließt Ziele hinter einer HTTP-Basic-Auth-Herausforderung auf Netzwerkebene ein (`--http-credentials`, v1.21.0) — bestätigt gegen eine echte, Caddy-geschützte Admin-Oberfläche, nicht nur eine synthetische Fixture: die gespeicherten Credentials beantworteten die Herausforderung beim ersten Versuch.

Nicht als eingechecktes Szenario ausgeliefert — das Dashboard-Layout und die Namen der Datenquellen sind spezifisch für die Infrastruktur eines Betreibers, und ein synthetisches Äquivalent zu erfinden würde duplizieren, was die lokale Fixture in Fall 1 bereits für strukturelle Regression abdeckt — nicht für diese Art geführter, mehrstufiger Konfigurationsarbeit.

Fall 5 — eine Ticketing-Plattform durchgängig administrieren

Dinoer über mehrere Sitzungen hinweg eingesetzt, um eine echte, selbstgehostete Ticketing-Installation zu konfigurieren und zu betreiben — Event-Einrichtung, Ticketkategorien, eine eigene Domain und die Scan-/Check-in-Werkzeuge am Veranstaltungstag — über dieselbe Weboberfläche, die ein menschlicher Administrator nutzen würde. Unterwegs traten reale Friktionen auf und wurden gelöst (Sitzungsverwaltung, Dropdown-Eigenheiten, ein Berechtigungs-Prompt, der einen unbeaufsichtigten Schritt blockierte) — keine reibungslose Erfolgsgeschichte, was Teil dessen ist, was sie zu einem nützlichen Beispiel macht: Die Hindernisse waren gewöhnliche Web-Automatisierungshindernisse, nichts Dinoer-Spezifisches.

Nicht als eingechecktes Szenario ausgeliefert — eine Ticketing-Konfiguration berührt Abrechnungs- und Veranstaltungsortdetails, die für den Betreiber spezifisch sind, dieselbe Begründung wie Fall 2.

Fall 6 — einen regionalen Veranstaltungskalender verfolgen

Eine einfache semantische Sondennutzung: einen Agenten bitten, einen lokalen Veranstaltungskalender auf bevorstehende Termine zu prüfen, ohne vorab zu wissen, welche Seite die Antwort enthält. Der schreibgeschützte Modus von Dinoer, kombiniert mit dem Accessibility-Baum, lässt den Agenten in einer Handvoll Anfragen scannen und zurückmelden — kein Vision-Modell für diese Art textgetriebener Aufgabe nötig. Eine Sitzung produzierte auch ein sauberes, reales Beispiel für das dokumentierte Falsch-positiv-Verhalten des WAF-Signals: eine Seite lud normal (reicher Inhalt, kein Captcha, kein Interstitial), während `respect.waf_bloquants` trotzdem auslöste — wegen einer nicht damit zusammenhängenden Drittanbieter-Ressource auf der Seite, die einem Erkennungsschlüsselwort entsprach — in etwa einer Minute gelöst, indem der bereits in derselben Antwort vorhandene Accessibility-Baum gelesen wurde, genau wie es die Regel „Signal, nie Verriegelung" des Leitfadens vorwegnimmt.

Nicht als eingechecktes Szenario ausgeliefert — eine bestimmte regionale Veranstaltungs-Site ist kein sta-

biles, reproduzierbares öffentliches Ziel, und eine namentlich zu nennen ist die Entscheidung des Betreibers, kein Projektstandard.

Fall 7 — realen Zugriff auf E-Commerce-Sites unter respektvoller Navigation testen

Eine wiederkehrende, ehrliche Beobachtung aus echten Sitzungen: respektvoll eingesetzt (ratenbegrenzte Verzögerungen, Seiten-/Aktionsobergrenzen, `--stealth` aktiv, kein Versuch, einen echten Block zu erzwingen), stellt Dinoer bei einer Reihe von E-Commerce-Sites fest, dass ein großer Anteil großer Plattformen einen glatten Block zurückgibt — HTTP 403, oder eine Anfrage, die nie abschließt — unabhängig davon, wie höflich der Traffic ist. Dies ist kein zu behebendes Dinoer-Manko: Anti-Bot-Haltung ist die eigene Entscheidung der Site, und Dinoer versucht nicht, sie zu überwinden (siehe „Respektvolle Navigation" oben). Praktisch: Für Preisvergleichsaufgaben gegen große kommerzielle Plattformen einen nennenswerten Anteil an Sackgassen erwarten, und ein Block-Signal (`respect.waf_bloquants`) als Information zum Umfahren behandeln, nicht als Fehler, gegen den erneut versucht werden sollte.

Eine Unterscheidung, die es sich zu merken lohnt: ein unsichtbarer Verifizierungsbildschirm, der nie aufgelöst wird und nichts zum Handeln bietet (keine Checkbox, keine Bildherausforderung), unterscheidet sich von einem interaktiven CAPTCHA. Letzteres ehrlich zu beantworten ist legitim — ein Agent, der für einen namentlich genannten Menschen von dessen eigener IP aus operiert, ist nicht der „Roboter", auf den die Frage abzielt. Ersteres bietet der Agentenseite schlicht keine Tür zum Öffnen, und es gewaltsam zu umgehen (IP-Rotation, TLS-Fingerprint-Fälschung) liegt außerhalb dessen, was Dinoer tut.

Bewusst nicht als eingechecktes Szenario ausgeliefert, und bewusst ohne Nennung der beteiligten Plattformen — siehe die WAF-Fragilitätsbegründung unter Fall 2: eine datierte Block-/Kein-Block-Tabelle, die an namentlich genannte kommerzielle Sites gebunden ist, veraltet und untergräbt ihren eigenen Punkt schneller, als sie ihn veranschaulicht. `docs/RETOUR_EXPERIENCE.md` FR-77 dokumentiert dasselbe Muster im Panel-Maßstab (39 % Sofort-Block-Rate).

Voraussetzungen vor dem Start

```
# 1. Prüfen, dass Dinoer antwortet
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y
# → muss {"succes": true, ...} zurückgeben

# 2. Prüfen, dass das verschlüsselte Verzeichnis gemountet ist (falls gocryptfs)
ls ~/Vaults/__PROJET__/Dinoer/
# → muss .json-Dateien zeigen, nicht verschlüsselten Inhalt

# 3. Credentials für eine Domain prüfen
/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-Konfiguration pro Projekt

Jedes Projekt kann sein eigenes Credentials-Verzeichnis haben. Zwei Methoden:

Methode 1 — direkte Umgebungsvariable (einmalig):

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

Methode 2 — projektspezifische `.dinoer.conf`-Datei (empfohlen für wiederkehrende Projekte):

```
# Datei im Projektwurzelverzeichnis erstellen
echo '{"secrets_dir": "../MyProject-secrets"}' > ~/git/MyProject/.dinoer.conf

# Dann jedem Aufruf voranstellen (oder zu Beginn der Shell-Sitzung exportieren)
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url …
```

Das `secrets_dir` in `.dinoer.conf` kann ein relativer Pfad sein — er wird relativ zum Speicherort der Datei `.dinoer.conf` aufgelöst.

Eine Seite erfassen und analysieren

```
# Seitenzustand lesen (schreibgeschützt)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y
# → gibt url_courante, titre_page, a11y_tree im JSON zurück
```

Was Sie erhalten: - `boussole.url_courante` + `boussole.titre_page`: effektive URL und Titel nach der Navigation - `a11y_tree`: Seitenstruktur als Text (Überschriften, Felder, Schaltflächen) - `etat.pret_a_agir` + `etat.raisons`: wahrgenommene Friktionen, damit das Modell sie umgeht

---

Ein Login-Formular automatisieren

Schritt 1 — Die Credentials-Datei vorbereiten.

Die Credentials-Datei heißt `<hostname>.json`, wobei `hostname` das Ergebnis von `urlparse(url).hostname` ist. Für `https://app.example.com/` lautet die Datei `app.example.com.json`.

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

Schritt 2 — Die Login-Seite erkunden.

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

`a11y_tree` lesen, um die Feld-Selektoren zu identifizieren.

Schritt 3 — Das Szenario schreiben.

```
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
```

Schritt 4 — Ausführen.

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

Mehrere Seiten in einem einzigen Aufruf validieren

Um N Seiten einer authentifizierten Site zu prüfen, ohne den Login jedes Mal zu wiederholen:

```
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
```

Einen Wert aus der Seite extrahieren

Um einen Textstring, einen Zähler oder einen beliebigen DOM-Wert zu lesen:

```
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
# → Ergebnis in evaluations[0].valeur
```

Für bereinigten dokumentarischen Text (Formulare und Rauschtags entfernt) stattdessen `extraire_texte` verwenden — die Ausgabe ist eine `titre`/`texte`/`url`/`date_capture`-Struktur, die ein zusammenfassender Agent direkt konsumieren kann.

Wichtig: JS-Skripte immer in eine `--actions`-Datei schreiben, nie inline mit `--action` (die Shell beschädigt verschachtelte Anführungszeichen).

Kontinuierliche strukturelle Überwachung einrichten (v1.18.0)

Dinoer hat keine visuelle Pipeline — Überwachung ist *strukturell*: Sie prüft den Statuscode der Seite, DOM-Elementzahlen und JS-Auswertungsergebnisse. Das ist günstiger als Bildvergleich und erfasst eine andere Klasse von Regression (zum Beispiel ein verschwundenes Formularfeld bei unverändertem Layout).

```
# 1. Einmalig eine strukturelle Referenz speichern
/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. Ein Prüf-und-Alarm-Durchlauf
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
```

Still wenn stabil, ein `ntfy` Push, wenn eine Regression erkannt wird. Planen Sie dies selbst mit cron – das Skript führt einen Durchlauf durch und beendet sich, es läuft nicht in einer Schleife. Im git-clone-Kanal wird `scripts/*.sh` niemals auf `/opt/dinoer/` bereitgestellt, daher wird der untenstehende Cron-Eintrag von der Git-Quelle ausgeführt, als Ihr eigener Benutzer (nicht das `dinoer` Dienstkonto, das keinen Zugriff auf `~/git/Dinoer/Dinoer/` hat). Im `.deb` Kanal werden die drei eingebetteten Skripte unter `/opt/dinoer/scripts/` installiert und sind über die `dinoer-*` Befehle erreichbar – korrigiert am 15. /08/2026, dieser Abschnitt ist vor der tatsächlichen Erstellung dieses Kanals entstanden:

```
# crontab -e (Ihre eigene 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
```

Häufige Stolperfallen

Situation	Was zu tun ist
`FileNotFoundError` bei der Credentials-Datei	Prüfen, dass die JSON-Datei mit dem vollständigen FQDN benannt ist (`urlparse(url).hostname`)
`SecretsFermesError` (Exit 42)	Das verschlüsselte Verzeichnis mounten: `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`
Ungültiges JSON in der Ausgabe	`2>/dev/null | tail -1` verwenden, um nur die JSON-Zeile zu extrahieren
Login gefolgt von Django-Redirect zum Dashboard	`naviguer` nicht in einer wiederaufgenommenen Django-Sitzung verwenden — die URL über `--url` übergeben
`<select>`-Formularfeld nicht ausgefüllt	`remplir` mit `selecteur` verwenden, dann auf die Option `cliquer`, oder über `evaluer` steuern
Klick hat keine Wirkung auf eine Schaltfläche außerhalb des sichtbaren Bereichs	`{"type":"defiler","selecteur":"#the-button"}` vor dem Klick hinzufügen
`auth_status: "active"` auch auf der Login-Seite	Positiver Selektor ist mehrdeutig (persistente Kopfzeile) — `--auth-indicator-negative .btn-login` hinzufügen
Web Components blockieren einen normalen Selektor	`cliquer_iframe`/`remplir_iframe` mit explizitem Selektor verwenden, oder über `evaluer` in die Shadow Root greifen
`respect.waf_bloquants` erscheint auf einer Seite, die tatsächlich nicht blockiert ist	Erkennung ist schlüsselwortbasiert (v1.16.0, verfeinert in v1.17.2) — als Signal behandeln, nicht als Urteil. Bleibt es auf einer bestätigt nicht blockierten Seite bestehen, `--ignorer-waf` hinzufügen
`cliquer` klickt auf das falsche Element einer Seite, die sich verändert hat	Reihenfolgestabile Selektoren bevorzugen, oder den Baum vor dem Klick mit einem frischen `--a11y`-Aufruf neu lesen
Ein langes RPA-Szenario schlägt auf halbem Weg fehl, und Sie wollen abgeschlossene Schritte nicht wiederholen	`--checkpoint DATEI` hinzufügen (v1.17.0) — denselben Befehl erneut starten, um fortzusetzen; DOM-Zustand bleibt nicht erhalten, nur Sitzung + Aktionsposition
Interaktive Elemente innerhalb eines iframes sind für den Baum unsichtbar	`cliquer_iframe`/`remplir_iframe` (v1.17.0) mit explizitem CSS-Selektor verwenden, oder `iframe_chemin` (v1.18.0) für ein in einem anderen verschachteltes iframe

Situation	Was zu tun ist
Ihr Modell meldet `"erreur": "guide_non_lu"` / Exit 1 beim ersten Dinoer-Aufruf	Erwartet beim ersten Mal, dass ein Modell Dinoer auf dieser Maschine als dieser Betriebssystembenutzer nutzt (v1.18.0) — es muss `docs/GUIDE_LLM.md` lesen und einmalig `--guide-version` übergeben. Das ist beabsichtigt, kein Fehler — dem Modell sagen, den Leitfaden zu lesen, statt den Fehler zu umgehen

---

Dinoer deinstallieren

Das Skript `~/git/Dinoer/Dinoer/scripts/uninstall.sh` entfernt die Installation sauber, in umgekehrter Reihenfolge zu `install.sh`.

```
# Zeigen, was entfernt würde, ohne etwas zu tun
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run

# Vollständige Deinstallation (interaktive Bestätigung)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh

# Ohne Bestätigung (Kalttests, verkettete Neuinstallation)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --confirme && bash
 ↪  ~/git/Dinoer/Dinoer/scripts/install.sh
```

Was entfernt wird:

Element	Detail
`/opt/dinoer/`	Code, Python-venv, Konfiguration
`/var/log/dinoer/`	Vorgangsprotokolle
Systembenutzer `dinoer`	Exklusiv für Dinoer erstellt
Systemgruppe `dinoer`	Dasselbe
Gruppenmitgliedschaft	Ihr Konto wird aus der Gruppe `dinoer` entfernt
Git-Pre-Push-Hook	`core.hooksPath` im Quell-Repository deaktiviert

Was nie angerührt wird: - `~/Vaults/` — Ihre Credentials - `~/git/Dinoer/` — Git-Quellen - Playwright-Browser-Cache (`~/.cache/ms-playwright/`)

Strukturierte Beweise (`/var/log/dinoer/preuves/`): Enthält das Verzeichnis Erfassungen, wird es standardmäßig mit einer Warnung beibehalten. Zum Entfernen:

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

Die Vorgangshistorie einsehen

```
# Alle Vorgänge zu einem Ziel
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local

# Nur mutierende Vorgänge (Klicks, Formulareingabe)
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local --mutatif

# Ab einem Datum
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local \
  --depuis 2026-06-01
```

Dinoer — Betriebshandbuch

Version 1.0.1 – September 2026

Dieses Dokument beantwortet eine Frage: **wie man X mit Dinoer macht**.

> **Wenn Sie Nutzer sind** — keine Befehle nötig. Sagen Sie Ihrem Modell, was Sie auf einer Website, einer Webanwendung oder einer Administrationsoberfläche besuchen, beobachten oder erreichen möchten. Das Modell liest dieses Handbuch und übersetzt Ihre Absicht in die richtigen Aktionen.
>
> **Wenn Sie ein Sprachmodell sind** — dies sind Ihre Befehle. Führen Sie sie direkt aus.

Keine Architekturbeschreibungen. Befehle, die funktionieren.

Inhaltsverzeichnis

---

1. Installation prüfen

```
# Günstigster möglicher Test – ohne Playwright, ohne URL, sofortiger Exit mit Code 0
↪  (v1.18.0+).
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --version
# → {"outil": "shot.py", "version": "1.0.1"}

# Vollständiger Test mit einem Befehl (~3 Sekunden).
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y --guide-version 1.6
```

Erwartetes Ergebnis: JSON auf stdout mit `"succes": true`.

`--guide-version` (v1.18.0+): `shot.py` und `rpa.py` verweigern die Ausführung ohne diesen Nachweis — außer ein lokaler Marker aus einem vorherigen akzeptierten Aufruf existiert bereits (`~/.config/dinoer/guide_state.json`). Der Wert ist das `<!-- notice-version: X.Y -->` in Zeile 3 von `docs/GUIDE_LLM.md` — nicht die Dinoer-Releasenummer. Lesen Sie den aktuellen Wert, statt einem hier zitierten Wert zu vertrauen: `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`. Siehe `docs/GUIDE_LLM.md`, Abschnitt „Mandatory pre-flight", für den vollständigen Mechanismus und das Fehlerformat, falls Sie ihn überspringen.

Sobald der Marker existiert, wird `--guide-version` wieder optional — jedes andere Befehlsbeispiel in diesem Handbuch lässt es absichtlich weg, da ein Marker aus einem beliebigen früheren erfolgreichen Aufruf sie bereits abdeckt, solange sich `notice-version` von `docs/GUIDE_LLM.md` seither nicht geändert hat.

```
# Überprüfen Sie die installierte Version.
```

```
grep "__version__" /opt/dinoer/shot.py
# → __version__ = "1.0.1"

# Überprüfen Sie, ob `playwright-stealth` verfügbar ist (Version v1.15.0).
/opt/dinoer/venv/bin/python -c "import playwright_stealth; print('stealth OK')"

# Überprüfen Sie, ob das verschlüsselte Verzeichnis gemountet ist.
ls ~/Vaults/__PROJET__/Dinoer/
# → müssen `.json`-Dateien anzeigen, keine leere Liste.
```

Gibt `ls ~/Vaults/...` eine leere Liste oder einen Fehler zurück: → mounten: `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`

1a. Installation

Zwei Kanäle, die sich gegenseitig ausschließen, wenn sie auf derselben Maschine verwendet werden.

`.deb` package — der normale Pfad, wenn Sie Dinoer unverändert verwenden möchten:

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

Pakete, Quellen und Prüfsummen werden auf dinoer.davalan.fr veröffentlicht. Die Konfiguration befindet sich unter `/etc/dinoer/dinoer.conf` (JSON, eine kommentierte Beispielkonfiguration ist daneben als `dinoer-sample.conf` installiert).

Git klonen – falls Sie den eigenen Code von Dinoer ändern möchten:

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

`scripts/install.sh` erstellt den Systembenutzer und die Gruppe `dinoer`, das Python- venv, deployt den Code auf `/opt/dinoer/`, installiert Chromium und führt einen Smoke- Test (`shot.py --a11y` gegen eine echte URL) aus. Deployen Sie weitere Änderungen mit `scripts/deploy.sh`.

Die Konfiguration befindet sich unter `/opt/dinoer/dinoer.conf` (JSON) auf diesem Kanal; der verschlüsselte Schlüssel für das Verzeichnis mit Geheimnissen ist `secrets_dir`. Projektbezogene Überschreibungen sind möglich über die Umgebungsvariable `DINOER_CONF` oder `~/.dinoer.conf`.

Deinstallation:

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run   # Vorschau, keine Änderungen
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh             # interaktive Bestätigung
```

---

2. Eine Seite lesen

2a. Schnelles Lesen — Text und Struktur, kein Bild

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

Liefert zurück: `a11y_tree` (Accessibility-Baum — die textuelle Struktur der Seite), `boussole` (effektive URL, Titel, HTTP-Status). Damit den Titel lesen, die URL prüfen oder die Seite kartieren, bevor interagiert wird.

2b. Bereinigter Seitentext

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

Liefert `extraction_texte` mit `titre`, `texte` (Rauschtags entfernt: `script`, `style`, `nav`, `header`, `footer`, `aside`, `noscript`), `url`, `date_capture`. Dies ist Dinoers Text der Seite — nie ein Screenshot.

2c. Zuerst die boussole lesen

Jede Ausgabe enthält ein `boussole`-Objekt — vor allem anderen lesen:

```
"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
  }
}
```

Stimmt `boussole.url_courante` nicht mit Ihrer Erwartung überein: vor jeder mutierenden Aktion anhalten und untersuchen.

2d. `etat` für eine Go/No-Go-Entscheidung lesen (v1.16.0)

Jeder erfolgreiche Lauf enthält ein `etat`-Objekt an der JSON-Wurzel — es vor jeder mutierenden Aktion lesen, statt `auth_status`, `respect.plafond_atteint`, `erreurs_js` und `erreurs_console` selbst manuell abzugleichen:

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

Ist `pret_a_agir` `false`: `raisons` auf die Ursache lesen (inaktive Authentifizierung, Sitzungsdrift, erreichte Navigationsobergrenze oder ein erkannter WAF-Block), bevor fortgefahren wird.

`etat` prüft nicht, ob URL oder Seiteninhalt Ihrer geschäftlichen Erwartung entsprechen — dafür `evaluer` mit `attendu`/`contient`/`motif` (Abschnitt 5d) verwenden.

---

3. Respektvolle Navigation (v1.15.0)

3a. Stealth-Modus `--stealth`

Manche Sites blockieren Headless-Browser bei `navigator.webdriver=true`, ohne die Absicht zu prüfen. `--stealth` entfernt diese automatische technische Markierung.

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

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

Wenn aktiv: `boussole.stealth_actif = true` in der JSON-Ausgabe.

Was `--stealth` verändert: `navigator.webdriver` wurden entfernt, Plugins, Sprachen und die Plattform wurden normalisiert. **Was `--stealth` nicht verändert**: Die IP-Adresse, Identität oder Navigationsabsicht des Nutzers.

3b. Höflichkeitsverzögerungen und Obergrenzen

Konfiguriert in `/opt/dinoer/dinoer.conf` (Abschnitt `[navigation]`). Standardwerte sind auch ohne Konfigurationsdatei aktiv (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`: Mindestverzögerung (ms) zwischen jeder Aktion. Mitgelieferter Standardwert: 800 ms.

Lokale Entwicklung — auf `0` setzen: Der Standardwert von 800 ms schützt einen unaufmerksamen Betreiber bei seinem *ersten, unkonfigurierten* Lauf gegen das öffentliche Internet — er hat keinen Schutzzweck gegen Ihre eigene Entwicklungsmaschine. Den Schlüssel explizit in Ihrer lokalen `dinoer.conf` setzen. Den Standardwert von 800 ms (oder höher) für jedes über das öffentliche Internet erreichte Ziel beibehalten.

Die Obergrenzen `max_pages_par_run` und `max_actions_par_run` stoppen den Lauf bei Überschreitung sauber — die Ausgabe-JSON enthält dann:

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

3c. Wirkungsmetriken

Jeder Lauf gibt `respect` zurück (JSON-Wurzel und innerhalb von `boussole`):

Schlüssel	Bedeutung
`pages_visitees`	Anzahl ausgeführter Navigationen vom Typ `naviguer`
`actions_executees`	Gesamtzahl ausgeführter Szenario-Aktionen
`duree_totale_ms`	Gesamtdauer des Laufs
`plafond_atteint`	`"max_pages_par_run"` oder `"max_actions_par_run"` bei vorzeitigem Stopp
`indice_agressivite`	Verhältnis mutierender Aktionen zur Gesamtzahl — bei offener Exploration unter 0,3 halten
`waf_bloquants`	Anzahl als WAF-blockiert markierter Navigationen

3d. Stealth-Benchmark — quantitativ (v1.17.1)

Konkrete Fingerprint-Signale zählen statt visuell zu vergleichen — dies ist die Methode, mit der der API-Kompatibilitätsfix für `playwright-stealth` in v1.17.0 verifiziert wurde (`docs/RETOUR_EXPERIENCE.md` FR-79):

```
# Ohne 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"}]'
```

```
# Mit 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"}]'
```

Die drei Werte in `evaluations[].valeur` lesen: `navigator.webdriver` sollte von `true` zu `false` wechseln, `td.failed` sollte gegen `0` fallen. Referenzmessung (v1.17.0-Fix, Sitzung 47): 12 fehlgeschlagen → 0 fehlgeschlagen.

3e. WAF-Erkennungssignal (v1.16.0, verfeinert in v1.17.2)

Dinoer markiert einen wahrscheinlichen WAF-Block passiv — HTTP 403/429 oder eine Übereinstimmung eines Titel-/HTML-Schlüsselworts (`Cloudflare`, `CAPTCHA`, `checking your browser` usw.). Dies ist ein Signal, nie eine Ausnahme — der Lauf wird normal abgeschlossen:

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

Ist es vorhanden und `> 0`: `etat.niveau_confiance` ist `"faible"` und `etat.pret_a_agir` ist `false`. Entscheiden Sie selbst, ob Sie mit `--stealth` erneut versuchen, das Ziel wechseln oder stoppen — Dinoer bricht den Lauf nicht für Sie ab.

Seit v1.17.2 stimmen generische Herstellernamen (`Cloudflare`, `Akamai`) nur mit dem Seitentitel überein — der Abgleich mit dem vollständigen HTML führte zuvor zu Falsch-positiven bei gewöhnlichen CDN-Ressourcenreferenzen. Bleibt ein Falsch-positiv bestehen, degradiert `--ignorer-waf` `niveau_confiance`, ohne `pret_a_agir: false` zu erzwingen (`boussole.waf_ignore_actif: true` protokolliert die Außerkraftsetzung). Die Erkennung ist schlüsselwortbasiert und kann Falsch-positive auf Seiten erzeugen, die legitim über Blockierung/ Erkennung diskutieren — als schnelles Signal behandeln, nicht als sicheres Urteil.

---

4. Verschlüsseltes Verzeichnis und Credentials

4a. Struktur

Die Credentials leben in einem verschlüsselten Verzeichnis — einem gocryptfs-Volume — mit einer `.json`-Datei pro Domain.

```
~/Vaults/__PROJET__/Dinoer/
  ├── app.example.com.json          ← Credentials für https://app.example.com/
  ├── admin.example.com.json        ← Credentials für https://admin.example.com/
  └── operations.jsonl              ← Vorgangsprotokoll (v1.15.0)
```

Format der Credentials-Datei:

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

Der Dateiname = `urlparse(url).hostname`. Für `https://app.example.com/login/`, erstellen Sie `app.example.com.json`. Das Verzeichnis stammt vom Schlüssel `secrets_dir` in der Konfigurationsdatei mit dem Namen `DINOER_CONF` (Standard: `/opt/dinoer/dinoer.conf`) – korrigiert am 15./08/2026: `~/.dinoer.conf` ist eine Namenskonvention für den Ort, an dem `DINOER_CONF` üblicherweise verweist,

und keine separate automatische Ausweichmaßnahme.

4b. Ein Formular ausfüllen — die absolute Regel

VERBOTEN — legt das Passwort in der Shell und `/proc` offen:

```
PASS=$(jq -r '.password' ~/Vaults/.../file.json)   # NIEMALS
curl -d "password=$PASS" https://...                 # NIEMALS
```

KORREKT — Credentials innerhalb von Playwright aufgelöst:

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

Werte gelangen nie durch die Shell, die Bash-History, Prozessprotokolle oder eine Datei.

4c. Die Credentials-Datei für einen Lauf wählen

```
# Standard-Credentials-Verzeichnis (definiert in dinoer.conf > secrets_dir)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y

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

# Projektspezifisches Credentials-Verzeichnis über .dinoer.conf
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y
```

Inhalt von `~/git/MyProject/.dinoer.conf`:

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

Der Pfad wird relativ zum Speicherort von `.dinoer.conf` aufgelöst.

Inhalt der `--secrets`-Datei — `origines_autorisees` seit dem 05/08/2026 zwingend (breaking change, keine Übergangsfrist): eine Datei ohne diesen Schlüssel wird vor jedem Lesevorgang zurückgewiesen.

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

`origines_autorisees` listet die Hostnamen, gegen die diese Datei verwendet werden darf — dasselbe Format wie `domaine_depuis_url()`: Kleinbuchstaben, kein Schema, kein Port. Ein Lesevorgang gegen eine Seite, deren Domain nicht in der Liste steht, wird verweigert (`SecretsOrigineNonAutoriseeError`).

4d. TOTP / MFA

Zwei aktive Pfade, beide innerhalb von Playwright aufgelöst (nie ein eingetippter Code):

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

Liest den Schlüssel `totp_cle` (Base32-Seed) aus der Credentials-Datei und berechnet den aktuellen TOTP-Code.

Um den Code über ntfy zu erhalten (Workflow ohne menschliches Eingreifen):

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

`selecteur` ist der CSS-Selektor des OTP-Felds. Die ntfy-Basis-URL kommt aus `DINOER_NTFY_URL` (Umgebungsvariable) oder dem Schlüssel `ntfy.url` von `dinoer.conf`.

4e. Integritätsprüfsumme (opt-in, v1.15.0)

Um eine Credentials-Datei gegen stille FUSE-Korruption zu schützen, ein `checksum`-Feld hinzufügen:

```
# Prüfsumme erzeugen
/opt/dinoer/venv/bin/python -c "
import json, hashlib
creds = json.load(open('my_credentials.json'))
# Korrigiert am 15/08/2026, gegen lib/repertoire_chiffre.py:32 geprüft
# (_CHAMPS_CHECKSUM): die Prüfsumme deckt alle vier vorhandenen Felder ab,
# nicht nur username/password.
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())
"
```

Den zurückgegebenen Wert zur Credentials-Datei hinzufügen:

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

Stimmt die Prüfsumme nicht überein, wirft `shot.py` `SecretsChecksumError` (Exit 42) mit einer expliziten Meldung. Ohne den Schlüssel `checksum`: unverändertes Verhalten (striktes Opt-in).

4f. Verschlüsseltes Verzeichnis geschlossen — was zu tun ist

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

```
# Das verschlüsselte Verzeichnis mounten
bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh

# Das Mounten prüfen
ls ~/Vaults/__PROJET__/Dinoer/
# → muss JSON-Dateien zeigen
```

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

Für Ziele hinter einer HTTP-Basic-Auth-Herausforderung auf Netzwerkebene (RFC 7617) — der Wall, den ein Reverse-Proxy wie Caddy, nginx oder Traefik errichtet, bevor überhaupt eine Seite gerendert wird, häufig vor selbstgehosteten Admin-Oberflächen. Dies ist ein anderer Mechanismus als die formularbasierte Authentifizierung oben (4a–4f), die vollständig unterstützt und unberührt bleibt.

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

Credentials-Datei — dasselbe einfache `username`/`password`-Paar, das bereits für den häufigen Fall genutzt wird (ein einziger Credential-Satz für das Ziel):

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

Die dedizierten Schlüssel `http_username`/`http_password` werden zuerst versucht und nur benötigt, wenn dasselbe Ziel *sowohl* einen Basic-Auth-Wall auf Netzwerkebene *als auch* ein eigenes, separates Anwendungs-Login hat (zwei verschiedene Credential-Paare in derselben Datei) — Dinoer fällt automatisch auf `username`/`password` zurück, wenn die dedizierten Schlüssel fehlen.

Bestätigt im Produktivbetrieb gegen ein echtes, Caddy-geschütztes Ziel: der sichere Standardwert (`send: "unauthorized"` — Credentials werden erst nach einem echten 401 gesendet, nie vorbeugend) löste die Herausforderung beim ersten Versuch. `boussole.http_credentials_actif: true` bestätigt einen echten Erfolg, nicht nur, dass das Flag übergeben wurde; `boussole.http_auth_requise: true` markiert einen ungelösten 401 getrennt von einem WAF-Block.

5. Ein RPA-Szenario schreiben und ausführen

5a. 3-Schritte-Protokoll

Schritt 1 — Die Seite erkunden (schreibgeschützt)

```
# Schnelle Ansicht — Accessibility-Baum
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y

# Vollständiges Lesen — Baum + bereinigter Text
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y \
  --action '{"type": "extraire_texte"}'

# Angereichertes DOM-Inventar (Frameworks, stabile data-Attribute)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/diagnostic_dom.json \
  --url https://target.local/
```

Worauf zu achten ist: - Stabile Attribute: `name`, `id`, `aria-label`, `data-testid` - Blockierende Overlays (Cookie-Banner, Modals) - SPA oder vollständiges HTTP-Reload

Schritt 2 — Das Szenario schreiben

```
{
  "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"}
  ]
}
```

Schritt 3 — Ausführen

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

5b. Vollständiges Szenario: einloggen und zwischen Seiten navigieren

```
{
  "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. Daten aus dem DOM extrahieren

```
{
  "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"}
  ]
}
```

Ergebnis 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 bei evaluer (nur rpa.py)

Drei sich gegenseitig ausschließende Schlüssel — einer pro Aktion:

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

Schlüssel	Vergleich	Gültige Typen
`attendu`	strikte Gleichheit `==`	str, int, bool
`contient`	Teilstring `in`	nur str
`motif`	`re.search()` Python	nur str

Schlägt die Assertion fehl: rpa.py stoppt sofort (Exit 1), bevor eine folgende mutierende Aktion ausgeführt wird.

5e. Unterszenarien (declencher_scenario)

Ein Login als wiederverwendbares Unterszenario definieren:

```
{
  "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"}
  ]
}
```

Dieses Unterszenario aus einem anderen Szenario aufrufen:

```
{
```

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

Maximale Tiefe: 5 Verschachtelungsebenen. `declencher_scenario` wird von `rpa.py` flach ausgerollt, bevor die Aktionen `shot.py` erreichen.

5f. Vor jeder Mutation prüfen, auf der richtigen Seite zu sein

Immer eine Absicherung als erste Aktion in Szenarien hinzufügen, die löschen oder ändern:

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

Schlägt die Absicherung fehl: rpa.py stoppt, bevor die Löschung ausgeführt wird.

5g. Eine Sitzung fortsetzen (persistierte Cookies)

```
# Erster Aufruf — authentifizieren und die Sitzung speichern
/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

# Folgeaufrufe — die Sitzung wiederverwenden (kein erneuter Login)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/dashboard/ \
  --reprendre-session /tmp/dinoer/session.json
```

Sitzungsdrift-Signal: Ist die Sitzung abgelaufen, `boussole.session_derive: true` im JSON. In diesem Fall: den vollständigen Login ohne `--reprendre-session` neu starten.

5h. Strukturelle Nicht-Regression — `--replay-verifier` (v1.17.0)

```
# Erster Lauf — die strukturelle Referenz speichern
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/dashboard.json \
  --sauver-verifier-reference /tmp/dashboard.ref.json

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

Vergleicht `http_status`, `dom_stats` und `evaluer`-Ergebnisse mit der gespeicherten Referenz. Urteil auf stderr:

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

Exit 1 bei `verdict: "regression"`, mit `diffs`, das jedes abweichende Feld auflistet (`reference` vs. `obtenu`). Die beiden Flags schließen sich gegenseitig aus.

5i. Ein langes Szenario nach einem Fehler fortsetzen — `--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
```

Schlägt das Szenario auf halbem Weg fehl, wird die Checkpoint-Datei mit der Anzahl abgeschlossener Aktionen und einer Sitzungsdatei geschrieben. **Denselben Befehl erneut starten**, um fortzusetzen: bereits abgeschlossene Aktionen werden übersprungen. Bei vollem Erfolg wird die Checkpoint-Datei automatisch gelöscht.

Ein durch eine Navigationsobergrenze (`max_actions_par_run`/ `max_pages_par_run`) gestoppter Lauf wird seit v1.17.2 genauso behandelt wie ein teilweiser Fehler — der Checkpoint wird mit dem tatsächlichen Fortschritt aktualisiert, nicht gelöscht.

Der DOM-Zustand (offene Modals, halb ausgefüllte Formulare) bleibt über eine Fortsetzung hinweg nie erhalten — nur Cookies/`localStorage` und die Position in der Aktionsliste. Sich nicht darauf verlassen, dass `--checkpoint` mitten in einem einzelnen mehrstufigen Formular fortsetzt; es setzt nur an Aktionsgrenzen fort.

5j. Elemente innerhalb eines iframes ansprechen (v1.17.0)

Innerhalb eines iframes (gleicher Ursprung oder Cross-Origin) existiert keine Elementnummerierung — direkt per CSS-Selektor ansprechen:

```
{"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` unterstützt `valeur: "depuis_secrets"` genau wie `remplir` (Abschnitt 4b) — nie ein Credential im Klartext im Szenario. Verweigert das Zielelement die Interaktion (z. B. ein `contenteditable`-Bereich im schreibgeschützten Zustand), `"force": true` zu `cliquer_iframe` hinzufügen — dieselbe Semantik wie `cliquer` (Abschnitt 7e).

Um den inneren Selektor zu finden: `evaluer` auf dem Inhalt des iframes nutzen, wenn er gleichen Ursprungs ist (`document.querySelector('iframe').contentDocument...`), oder bei Cross-Origin das eigene Markup/die eigene Dokumentation der Zielanwendung konsultieren.

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

Ein iframe innerhalb eines anderen iframes: `iframe_selecteur` durch `iframe_chemin` ersetzen, ein geordnetes Array — ein CSS-Selektor pro Verschachtelungsebene, von außen nach innen.

```
{"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` (einzelner Frame) und `iframe_chemin` (verschachtelter Abstieg) schließen sich gegenseitig aus — genau einer ist pro Aktion erforderlich. Für ein einstufiges iframe weiterhin `iframe_selecteur` verwenden (Abschnitt 5j).

---

6. Aktionen — vollständige Referenz

Typ	Erforderliche Parameter	Optionale Parameter	Hinweise
`naviguer`	`url`	—	Vollständiges HTTP-Reload. Wird in `respect.pages_visitees` gezählt

Typ	Erforderliche Parameter	Optionale Parameter	Hinweise
`cliquer`	`selecteur`	`force` (bool), `repli_js` (bool)	`force: true` umgeht CSS-verborgene Elemente oder showModal. `repli_js: true` versucht bei fehlgeschlagenem nativem Klick erneut über JS (v1.22.0) — mit `--no-evaluer` abgelehnt (Exit 2, vor dem Start)
`remplir`	`selecteur`, `valeur`	`secret_cle`	`valeur: "depuis_secrets"` erfordert `secret_cle`; `"depuis_secrets_totp"` für TOTP
`evaluer`	`script`	`attendu`, `contient`, `motif`	Im Browser ausgeführtes JS. Assertions nur für rpa.py
`defiler`	`px` oder `selecteur`	—	Vertikales Scrollen in Pixeln (`px`) oder Scrollen zum Element (`selecteur`)
`pause`	`ms`	—	Feste Verzögerung in ms. `attendre_selecteur_present` für DOM-Signale bevorzugen
`attendre`	`selecteur`	—	Wartet, bis der CSS-Selektor sichtbar wird (`state=visible`, Playwright-Standard — korrigiert am 16/08/2026, identisch zu `attendre_selecteur_present`)
`attendre_navigation`	—	—	Wartet auf `networkidle` (Ende der Netzwerkanfragen)
`attendre_url`	`motif`	`attendre_changement` (bool)	URL-Teilstring-Abgleich. `attendre_changement: true` wartet zuerst auf eine echte Navigation (siehe die FR-55-Falle)
`attendre_selecteur_present`	`selecteur`	—	Wartet, dass das Element sichtbar ist (`state=visible`)
`attendre_absence`	`selecteur`	`delai_initial_ms`	Wartet auf Entfernung des Elements aus dem DOM (`state=detached`)
`attendre_reseau_calme`	—	`timeout_ms`	500 ms Netzwerkstille. `timeout_ms`: maximale Dauer, bevor aufgegeben wird
`attendre_mfa_ntfy`	`selecteur`	`timeout`	Wartet auf einen TOTP-Code über ntfy, füllt ihn in das Feld
`nettoyer_overlay`	`selecteur`	—	Verbirgt blockierende Overlays (Cookie-Banner, Modal) — expliziter Selektor, keine automatische Erkennung

Typ	Erforderliche Parameter	Optionale Parameter	Hinweise
`declencher_scenario`	`scenario`	—	Fügt die Aktionen eines Unterszenarios ein. Maximale Tiefe: 5 (rpa.py)
`extraire_texte`	—	—	Bereinigter Seitentext aus dem gerenderten DOM — `extraction_texte` (`titre`, `texte`, `url`, `date_capture`)
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force` (bool)	Klick innerhalb eines iframes (v1.17.0). `iframe_chemin` für verschachtelte iframes (v1.18.0, Abschnitt 5k)
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`	Ausfüllen innerhalb eines iframes (v1.17.0). `valeur: "depuis_secrets"` unterstützt

---

7. Häufige Hindernisse behandeln

7a. Cookie-Banner / blockierendes Overlay

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

Vor jeder anderen Lese-/Interaktionsaktion platzieren. Das Overlay maskiert Elemente im Accessibility-Baum.

7b. Element außerhalb des sichtbaren Bereichs

Dorthin scrollen (nach Betrag oder nach Selektor), dann handeln:

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

oder

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

7c. SPA (React, Vue, Angular) — navigieren ohne Reload

Nach einem Klick, der die Ansicht in einer SPA ändert, weiß Playwright nicht, wann die Navigation abgeschlossen ist.

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

Nie annehmen, dass ein Klick die Navigation ohne DOM-Signal abgeschlossen hat. Nach einem Submit `attendre_url` mit `attendre_selecteur_present` kombinieren (Teilstring-Abgleich-Falle, siehe `docs/GUIDE_LLM_INTERACTIONS.md`).

7d. CSS-Dialog oder showModal()

`TimeoutError` bei `cliquer`, obwohl das Element im DOM sichtbar ist = CSS-verborgenes Element oder innerhalb eines Dialogs.

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

Reicht `force: true` nicht aus (Interaktionsfähigkeits-/Obstruktionsfehler): `repli_js: true` zur selben Aktion hinzufügen (v1.22.0), oder auf JS zurückgreifen:

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

7e. Lange Operation (Spinner, Batch-Job)

`pause` nicht verwenden, um eine feste Dauer abzuwarten. Auf das DOM-Signal warten:

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

Liefert die Operation kein DOM-Signal, den Zustand mit `evaluer` abfragen und fortfahren, sobald der Beleg vorliegt.

7f. Obergrenze erreicht (v1.15.0)

Ist `respect.plafond_atteint` in der Ausgabe vorhanden, wurde der Lauf gestoppt, bevor das Szenario abgeschlossen war. Verbleibende Aktionen wurden nicht ausgeführt.

Optionen: 1. `max_pages_par_run` oder `max_actions_par_run` in `dinoer.conf` erhöhen 2. Das Szenario auf mehrere Läufe aufteilen 3. Einen Teilabschnitt mit `--checkpoint` fortsetzen

7g. `<select>`-Formularfeld

`remplir` (`.fill()`) funktioniert nicht bei `<select>`. Einen JS-Setter über `evaluer` verwenden:

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

7h. Von WAF blockierte Site (sofortiger 403)

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

Bleibt 403 mit `--stealth` bestehen: Die Site nutzt TLS-Fingerprinting (JA3/JA4) oder fortgeschrittene Verhaltensanalyse (Cloudflare Enterprise). `playwright-stealth` umgeht diese Schutzmaßnahmen nicht. Siehe `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 für den Kontext.

Dinoer markiert auch passiv einen wahrscheinlichen Block — siehe Abschnitt 3e (`respect.waf_bloquants`).

7i. Initiale Navigation schließt nie ab — `--wait-until` (v1.22.0)

Symptom: `TimeoutError` bei der initialen Navigation, und das Erhöhen von `--timeout` ändert nichts (45 s scheitert genauso wie 10 s). Ursache: Standardmäßig wartet Dinoer auf `networkidle` — 500 ms Netzwerkstille. Eine Seite, die kontinuierlich abfragt (Live-Statistiken, automatisch aktualisierende Zähler, Router-Admin-Panels), erzeugt diese Stille nie, sodass kein Timeout-Wert je groß genug sein kann.

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

# rpa.py — an shot.py weitergereicht, sodass Szenarien dieselben Ziele erreichen
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario ./admin_login.json --wait-until load
```

Ein Szenario kann es stattdessen als Wurzeleigenschaft tragen und dadurch in sich geschlossen bleiben:

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

Das CLI-Flag hat Vorrang vor der Szenario-Eigenschaft.

Wert	Wartet auf	Verwenden, wenn
`networkidle`	500 ms Netzwerkstille	Standard — beibehalten, sofern kein Fehlschlag
`load`	`load`-Ereignis (Seite und Unterressourcen)	kontinuierliches Abfragen / Live-Statistiken
`domcontentloaded`	HTML geparst, Unterressourcen noch ausstehend	sehr schwere Seite, nur der DOM wird benötigt

Gilt nur für die initiale Navigation — die Aktion `naviguer` ist unberührt. `boussole.wait_until` meldet den Wert nur, wenn er vom Standard abweicht.

---

8. Überwachung — strukturelle Prüfungen

In Dinoer existiert keine bildbasierte Überwachung (kein visueller Diff). Strukturelle Prüfungen sind textbasiert und CI-freundlich.

8a. Kontinuierliche strukturelle Überwachung — `scripts/monitor-verifier.sh` (v1.18.0)

Überwacht *Struktur* (`http_status`, `dom_stats`, `evaluations`) — null Bild, null LLM-Aufruf, aufgebaut auf `--replay-verifier` (Abschnitt 5h).

```
# Erster Durchlauf – Erstellung der strukturellen Referenz.
/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

# Ein einmaliger Check und Alarm – kein Hintergrunddienst, sondern ein Skript, das
 ↪  wiederholt über Cron ausgeführt wird.
# Auf dem git-clone Kanal werden Skripte aus dem Verzeichnis "scripts/*.sh" niemals auf
 ↪  /opt/dinoer/ bereitgestellt.
# Es wird also direkt aus dem Git-Quellcode ausgeführt, und zwar als dein eigener Benutzer.
 ↪  Im .deb-Kanal...
# die drei verpackten Skripte (monter/demonter-repertoire-chiffre,
# (Monitor-Verifizierungsmodul) sind unter /opt/dinoer/scripts/ installiert – korrigiert.
# 15/08/2026, dieser Kommentar stammt vor der eigentlichen Erstellung dieses Kanals.
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 (Ihre eigene 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
```

Stabil ⟶ Stille. Regression ⟶ eine `ntfy`-Benachrichtigung mit dem Diff. Jeder Aufruf ist ein isolierter Prozess — kein Daemon, kein Speicherleck-Risiko, und die Obergrenzen der respektvollen Navigation werden bei jedem Durchlauf sauber zurückgesetzt.

Korrigiert am 16/08/2026: Das Skript rief früher `rpa.py --no-capture --replay-verifier` auf, aber `--no-capture` war kein `rpa.py`-Flag mehr — jeder echte Aufruf scheiterte an argparse. Das tote Flag wurde entfernt (Dinoer hat standardmäßig keinen Bildpfad, das Entfernen ändert sonst nichts).

Nuance des Guide-Read-Locks: Wird es unter einem separaten Betriebssystembenutzer aufgerufen (z. B. ein Systemdienstkonto), muss dieser Benutzer `--guide-version` einmalig validiert haben (`~<home>/.config/dinoer/guide_state.json`).

---

9. Vorgangsprotokoll

Das Protokoll ist `/var/log/dinoer/operations.jsonl`. Liegt der konfigurierte Protokollpfad innerhalb des verschlüsselten Verzeichnisses und ist es nicht gemountet, werden Einträge zu einem lokalen Fallback umgeleitet (degradiertes Schreiben, 700/600), statt im Klartext auf dem rohen Host geschrieben zu werden.

```
# Die letzten 10 Einträge lesen
tail -n 10 /var/log/dinoer/operations.jsonl | python3 -m json.tool

# Nach Ziel filtern (journal.py-Werkzeug)
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com

# Nur mutierende Vorgänge filtern
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --mutatif

# Ab einem Datum
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --depuis 2026-07-01

# Nur fehlgeschlagene Läufe (v1.20.0) — resultat != success
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --erreurs
```

Felder in jedem Eintrag:

Feld	Bedeutung
`ts`	ISO 8601-Zeitstempel
`operation_id`	Eindeutige Laufwerkskennung (v1.16.0), benennt das temporäre Verzeichnis
`outil`	`"shot.py"`, `"rpa.py"` oder `"campagne.py"` — korrigiert am 15/08/2026, wurde als `mode` dokumentiert (`shot.py`/`rpa.py`), ein Feld, das der Code nie geschrieben hat
`version`	Dinoer-Version
`cible_url`	Ziel-URL
`resultat`	`"succes"` oder `"echec"`
`mutatif`	`true` wenn mindestens eine Schreiboperation durchgeführt wurde
`hostname_executant` / `utilisateur_executant` / `profil_actif`	Ausführungs-Kontext (Host, Betriebssystem-Benutzer, aktives Operatorprofil)
`intention`	Label, das über `--intention` oder das Szenariofeld `intention` übergeben wurde — nur vorhanden, wenn es gesetzt ist
`source_scenario`	Nur der Name der Szenariodatei, ohne Pfad (v1.18.0) — nur im RPA-Modus vorhanden. Korrigiert am 15/08/2026: zuvor enthielt die Tabelle auch ein Feld `scenario` (vollständiger Pfad); der Code hat keines davon geschrieben
`chainage`	Liste von `{scenario, profondeur, action_debut, action_fin}` — nur vorhanden, wenn `declencher_scenario` verwendet wurde

Feld	Bedeutung
`actions`	Zusammengefasste Aktionsliste — vorhanden, wenn der Lauf Aktionen enthielt
`actions_raw`	Neutralisierte vollständige Aktionsliste — nur bei einem erfolgreichen Lauf mit Aktionen vorhanden
`captures`	Referenz(en) zu strukturellen Erfassungen — nur vorhanden, wenn der Lauf irgendwelche Ergebnisse erzeugt hat
`erreur`	Nur beim Fehler vorhanden
`respect`	Das Navigationsprotokoll des Laufs — nur vorhanden, wenn es gesetzt ist
`evaluations`	Bereinigte `{script, valeur_retournee}` Werte — nur vorhanden, wenn `evaluer` Aktionen ausgeführt wurden

Es gibt kein Feld `duree_ms` im Journal — am 15/08/2026 korrigiert, diese Tabelle führte zuvor eines auf, das der Code nie geschrieben hat. Die Zeitmessung pro Aktion ist `latences_actions`, im JSON-Output eines Laufs, nicht im Journal-Eintrag.

9a. Log-Rotation (G-36)

Dinoer liefert keine logrotate-Konfiguration — `/var/log/dinoer/operations.jsonl` wächst unbegrenzt, bis der Administrator eine installiert. `lib/journal.py` öffnet und schließt die Datei bei jedem Schreibvorgang (kein persistenter Dateideskriptor über Läufe hinweg), genau damit das **Standard**-Verhalten von logrotate (aktuelle Datei umbenennen, eine neue erstellen) ohne jede besondere Option korrekt funktioniert: Der nächste Schreibvorgang öffnet den Pfad erneut und findet die neue Inode.

Kein `copytruncate` zu einer Dinoer-logrotate-Konfiguration hinzufügen — es ist hier unnötig und führt ein Zeitfenster für Schreibverlust wieder ein. Beispiel `/etc/logrotate.d/dinoer`:

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

`journal.py` (der Leser) folgt rotierten Dateien bereits transparent (`operations.jsonl`, `.1`, `.2.gz`, …) — nach der Rotation ist kein weiterer Schritt nötig.

---

10. CLI-Flags — Referenz

shot.py

Flag	Standard	Beschreibung
`--version`	—	Gibt die installierte Version aus und beendet sich sofort — kein Playwright, kein weiteres Argument erforderlich (v1.18.0)

Flag	Standard	Beschreibung
`--guide-version X.Y`	—	Nachweis, `docs/GUIDE_LLM.md` gelesen zu haben — erforderlich, außer ein gültiger lokaler Marker existiert bereits (v1.18.0, Abschnitt 1)
`--url URL`	erforderlich	zu erfassende URL
`--actions DATEI`	—	JSON-Datei mit sequenziellen Aktionen
`--action JSON`	—	Einzelne Aktion als Inline-JSON — sorgfältig quoten, bei JS-lastigen Aktionen `--actions DATEI` bevorzugen
`--attendre-selecteur SEL`	—	Vor Abschluss des Laufs auf einen Selektor warten
`--timeout MS`	10000	Playwright-Timeout pro Aktion (ms)
`--wait-until WERT`	`networkidle`	`networkidle`|`load`|`domcontentloaded` — nur initiale Navigation (v1.22.0, Abschnitt 7i)
`--largeur PX`	1280	Viewport-Breite
`--hauteur PX`	720	Viewport-Höhe
`--a11y`	aus	Accessibility-Baum im JSON einschließen
`--stealth`	aus	playwright-stealth-Modus (v1.15.0)
`--secrets DATEI`	—	expliziter Pfad zu einer Credentials-Datei
`--auth-indicator SEL`	—	CSS-Selektor, der nur in authentifizierter Sitzung vorhanden ist
`--auth-indicator-negative SEL`	—	erfordert `--auth-indicator`; CSS-Selektor, der nur außerhalb der authentifizierten Sitzung vorhanden ist
`--ignorer-waf`	aus	Ein erkannter WAF-Block degradiert `niveau_confiance`, erzwingt aber nicht mehr allein `pret_a_agir: false` (v1.17.2, Abschnitt 3e)
`--http-credentials`	aus	Löst HTTP-Basic-Auth-Credentials aus der Credentials-Datei auf, begrenzt auf den Ursprung des Ziels (v1.21.0, Abschnitt 4g)
`--ignore-tls-errors`	aus	Ungültiges TLS auf kontrollierten LAN-/Dev-Zielen akzeptieren — nie im öffentlichen Internet (v1.15.1)
`--no-evaluer`	aus	Verweigert die Aktion **evaluer** (und `repli_js`) für den gesamten Lauf — empfohlen bei sensiblen Formularen (v1.15.1)
`--no-filtre-evaluer`	aus	Deaktiviert die stdout-Neutralisierung von **evaluer**-Rückgabewerten, URLs und Fehlermeldungen — nur für explizite Debug-Läufe; deaktiviert wird `boussole.filtre_evaluer_actif: false` gesetzt (v1.23.0)
`--intention TEXT`	—	im Protokoll erfasstes geschäftliches Label
`--sauver-session DATEI`	—	speichert Cookies nach den Aktionen
`--reprendre-session DATEI`	—	setzt eine gespeicherte Sitzung fort

Flag	Standard	Beschreibung
`--source-scenario NAME`	—	intern (rpa.py-Verrohrung für das Protokoll — nicht für direkte Aufrufe)
`--chainage JSON`	—	intern (rpa.py-Verrohrung für das Protokoll — nicht für direkte Aufrufe)

rpa.py

Reicht alle relevanten shot.py-Flags weiter, plus:

Flag	Beschreibung
`--version`	gibt die installierte Version aus und beendet sich sofort (v1.18.0)
`--guide-version X.Y`	Nachweis, `docs/GUIDE_LLM.md` gelesen zu haben — unabhängig geprüft, dieselbe Regel wie shot.py (v1.18.0)
`--scenario DATEI`	Pfad zu JSON- oder YAML-Szenario (erforderlich)
`--url URL`	überschreibt die Szenario-URL, ohne die Datei zu ändern
`--stealth`	an shot.py weitergereicht
`--wait-until`	an shot.py weitergereicht (v1.22.0, Abschnitt 7i)
`--ignorer-waf`	an shot.py weitergereicht (v1.17.2, Abschnitt 3e)
`--http-credentials`	an shot.py weitergereicht; auch als Szenario-Wurzeleigenschaft `"http_credentials": true` setzbar (v1.21.0, Abschnitt 4g)
`--auth-indicator-negative`	erfordert einen `auth_indicator` (CLI oder Szenario-Wurzeleigenschaft)
`--sauver-verifier-reference DATEI`	speichert die strukturelle Referenz für `--replay-verifier` (v1.17.0, Abschnitt 5h)
`--replay-verifier DATEI`	vergleicht den Lauf mit einer strukturellen Referenz, Exit 1 bei Regression (v1.17.0, Abschnitt 5h)
`--checkpoint DATEI`	setzt ein langes Szenario nach einem Fehler mitten im Lauf fort (v1.17.0, Abschnitt 5i)

campagne.py (Recherche-Pipeline)

Flag	Beschreibung
`--manifeste DATEI`	Kampagnen-Manifest (JSON) — erfordert `id_campagne` + `cibles`
`--id-campagne ID`	Kampagnen-Kennung (im Manifest und der Extraktion verwendet)
`--extraire-cible ANFRAGE`	gezielte Extraktion auf einem bereits gesammelten Korpus, ohne Synthese. Erfordert `--id-campagne` oder `--corpus`
`--corpus DATEI`	direkter Pfad zu einer `collecte.jsonl` — Alternative zu `--id-campagne` für `--extraire-cible`
`--format-extraction {json,markdown,html}`	Ausgabeformat von `--extraire-cible` (Standard: `json`) — am 15/08/2026 hinzugefügt
`--desactiver-cache`	den Suchcache umgehen
`--purger-cache`	den gesamten Suchcache leeren
`--purger-cache-avant-jours N`	Cache-Einträge älter als N Tage leeren

Zieltypen im Manifest: `query`, `url`, `produit`, `table_reference`. Artefakte: das geteilte `/var/log/dinoer/operations.jsonl` + eine kampagnenspezifische `collecte.jsonl`. Vollständiges Detail: `campagne.py --help`.

11. Exit-Codes und Ausgabe

Exit-Codes

Code	Ursache	Was zu tun ist
0	Erfolg	—
1	Playwright-Fehler, fehlgeschlagene Aktion, rpa.py Assertion, `action_secret_en_clair`	Lesen Sie `erreur` im JSON. Sehen Sie sich `GUIDE_LLM_INTERACTIONS.md` an.
1	`guide_non_lu` – Fehlende/falsche `--guide-version`, kein gültiger Marker (v1.18.0)	Wird vor dem Start von Playwright ausgelöst. Lesen Sie `docs/GUIDE_LLM.md`, starten Sie neu mit `--guide-version X.Y` (Abschnitt 1).
2	Inkompatible Argumente, `arguments_incompatibles`, `url_scheme_interdit`, `chemin_sensible_refuse`	Lesen Sie `message` – es benennt den Konflikt.
3	Modul `playwright` nicht gefunden	Aufrufen über `/opt/dinoer/venv/bin/python`.
42	`SecretsFermesError` – Verschlüsseltes Verzeichnis nicht gemountet oder ungültige Prüfsumme	Mounten Sie es, oder überprüfen Sie die Anmeldedatei.
43	`SecretsNonConfigureError` – Kein `secrets_dir` konfiguriert	Konfigurieren Sie `secrets_dir` in `dinoer.conf` (undo ein fehlendes Beispiel: erstellen Sie `/opt/dinoer/dinoer.conf`).

Struktur der Ausgabe-JSON

```
{
  "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) ist immer vorhanden und identifiziert diesen Lauf eindeutig — er stimmt mit dem `operation_id`-Feld des Eintrags dieses Laufs im Vorgangsprotokoll überein (Abschnitt 9) und benennt das Belegverzeichnis `preuves/<AAAA-MM>/<operation_id>/`, wenn Aufnahmen dort archiviert werden (korrigiert am 15/08/2026, gegen `lib/journal.py` geprüft: nirgends im Code existiert ein Verzeichnis `/tmp/dinoer/<operation_id>/` — `/tmp/dinoer/` enthält immer nur die Ausweich-Protokolldatei des Vorgangsprotokolls). `etat` (v1.16.0) ist nur auf dem Erfolgspfad vorhanden. `latences_actions` (v1.20.0) ist immer vorhanden (leere Liste ohne Aktionen), ein Eintrag pro tatsächlich abgesetzter Aktion — siehe `GUIDE_LLM_MONITORING.md`, wie es `respect.duree_totale_ms` ergänzt.

Bedingte Schlüssel (fehlen, wenn inaktiv): `dom_stats`, `a11y_tree`, `evaluations`, `extraction_texte`, `auth_status`, `stealth_actif`, `session_derive`, `respect.plafond_atteint`, `respect.waf_bloquants`, `respect.indice_agressivite` (vorhanden, sobald mindestens eine Aktion lief), `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` (nur Fehler-JSON, v1.17.0). Siehe `GUIDE_LLM_MONITORING.md` für die vollständige Aktivierungstabelle.

Fehler — 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": "" }
}
```

Referenzpfade

Pfad	Rolle
`/opt/dinoer/`	Produktivinstallation
`/opt/dinoer/venv/bin/python`	für jeden Aufruf zu verwendendes Python
`/opt/dinoer/dinoer.conf`	Maschinenkonfiguration (secrets_dir, navigation, ntfy)
`/opt/dinoer/scenarios/`	RPA-Szenarien (einschließlich `diagnostic_dom.json`)
`/opt/dinoer/docs/`	Dokumentation
`/opt/dinoer/references/`	Referenzen für `--sauver-verifier-reference` / Replay
`/tmp/dinoer/`	Sitzungsdateien (`--sauver-session`/`--reprendre-session`) und die Ausweich-Protokolldatei — korrigiert am 15/08/2026: kein Unterverzeichnis pro `operation_id` hier, siehe nächste Zeile
`/var/log/dinoer/preuves/<AAAA-MM>/<operation_id>/`	Belegverzeichnis für einen Lauf, isoliert nach `operation_id` (v1.16.0), nur erstellt, wenn Aufnahmen archiviert werden
`~/Vaults/__PROJET__/Dinoer/`	Credentials + Protokoll (gocryptfs-Volume)
`~/git/Dinoer/Dinoer/`	Git-Quellen (hier bearbeiten, dann `deploy.sh`)
`/var/log/dinoer/operations.jsonl`	persistentes Vorgangsprotokoll (`journal.py`)

Nach Änderung der Quellen deployen:

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