OpenClaw Security

Dieser Bericht analysiert Sicherheitskonzepte und -muster im OpenClaw-Repo, mit Schwerpunkt auf Zugangsdaten-Sicherheit, Konfigurierbarkeit und Prompt Injection. Quellen:

- `docs/gateway/security/index.md`
- `docs/gateway/sandboxing.md`
- `docs/gateway/sandbox-vs-tool-policy-vs-elevated.md`
- `docs/gateway/authentication.md`
- `docs/gateway/pairing.md`
- `src/security/external-content.ts`
- `src/cron/isolated-agent/run.ts`

1) Sicherheitsmodell auf hoher Ebene

OpenClaw versteht Sicherheit als Schichten von Kontrollen, die den Schadensradius reduzieren, statt "perfekte Prävention" zu versprechen. Das Modell:

1. **Zugriffskontrolle zuerst**: wer den Bot ansprechen darf (DM-Pairing, Allowlists, Gruppenrichtlinien).
2. **Scope-Kontrolle**: welche Tools aktiviert sind und wo sie laufen dürfen (Tool-Policy + Sandboxing + Elevated-Exec-Gates).
3. **Modellauswahl und Prompts**: Leitplanken verstärken, aber nicht allein darauf verlassen.

Das spiegelt sich explizit in `docs/gateway/security/index.md`, wo eingehender Zugriff und Tool-Schadensradius höher gewichtet werden als Modellstärke, sowie in den Sandbox/Tool-Policy-Dokumenten, die "wo Tools laufen" von "welche Tools erlaubt sind" trennen.

2) Speicherung von Zugangsdaten und Expositionsflächen

OpenClaw speichert mehrere Arten von Zugangsdaten auf der Platte unterhalb des State-Verzeichnisses (Standard: `~/.openclaw`). Das Security-Dokument nennt u. a.:

- Channel-Zugangsdaten und Allowlists unter `~/.openclaw/credentials/**`.
- Model-Auth-Profile unter `~/.openclaw/agents/<agentId>/agent/auth-profiles.json`.
- Session-Transkripte unter `~/.openclaw/agents/<agentId>/sessions/*.jsonl`.

Diese Dateien gelten als sensibel, und der Security-Audit (`openclaw security audit --fix`) zieht die Rechte auf 700 für das State-Verzeichnis und 600 für Konfigurations-/Credential-Dateien an (`docs/gateway/security/index.md`).

Das **State-Verzeichnis ist konfigurierbar** über `OPENCLAW_STATE_DIR`. Es in einen dedizierten, gesperrten Pfad zu verschieben, ist die wichtigste eingebaute Methode, Zugangsdaten zu "verstecken", ohne Code zu ändern (`docs/gateway/security/index.md`, `docs/gateway/pairing.md`).

3) Tool-Policy, Sandboxing und Elevated Exec

OpenClaw hat eine dreiteilige Kontrollfläche (siehe `docs/gateway/sandbox-vs-tool-policy-vs-elevated.md`):

- **Sandbox**: wo Tools laufen (Docker-Container vs Host). Gesteuert durch `agents.defaults.sandbox.*` und per-Agent-Overrides.
- **Tool-Policy**: welche Tools aufrufbar sind (Allow/Deny, Profile, provider-spezifische Controls).
- **Elevated**: ein reiner Exec-Escape-Hatch, um Befehle auf dem Host auszuführen, wenn gesandboxed.

Wichtige Defaults und Security-Implikationen (`docs/gateway/sandboxing.md`):

- `workspaceAccess: "none"` hält den Host-Workspace standardmäßig aus dem Container und nutzt einen sandbox-spezifischen Workspace.
- Bind-Mounts (`sandbox.docker.binds`) sind explizit und durchbrechen die Sandbox. Das Security-Dokument warnt vor dem Mounten von Secrets und empfiehlt `:ro`, wo nötig.
- Ist Sandboxing aus, laufen `exec`, `read`, `write` etc. direkt auf dem Host.

Fazit: Die primäre Sicherheitsgrenze ist, wie strikt Tools und Sandbox-Mounts begrenzt werden. Wenn Tools, die Dateien lesen oder Commands ausführen können, auf dem Host aktiviert sind, liegen alle Secrets im Zugriff des OS-Users im Scope.

4) Umgang mit Prompt Injection

OpenClaw behandelt externe Inhalte als untrusted und kapselt sie vor dem Senden an das Modell:

- `src/security/external-content.ts` definiert einen Security-Wrapper mit expliziten Warnungen und Boundary-Markern (z. B. `<<<EXTERNAL_UNTRUSTED_CONTENT>>>`).
- Er erkennt verdächtige Muster (z. B. "ignore previous instructions", "system prompt", "rm -rf") und loggt sie.
- `src/cron/isolated-agent/run.ts` nutzt diesen Wrapper standardmäßig für externe Hook-Inhalte, mit einem expliziten Escape-Hatch (`allowUnsafeExternalContent`).

Das ist **Defense in Depth**, kein vollständiger Schutz. Die Security-Dokumente betonen, dass Prompt Injection nicht durch System Prompts allein gelöst wird und mit Tool-Policy, Sandboxing und strikten Allowlists eingegrenzt werden muss.

5) Antworten auf die konkreten Fragen

Sind User-Credentials (.ssh Keys, Browser-Credentials, Cookies/Passwort-DBs) sicher?

SSH Keys

- OpenClaw selbst liest keine privaten SSH Keys. Es nutzt das System-`ssh` für Config-Auflösung und Tunneling (z. B. `src/infra/ssh-config.ts`, `src/infra/ssh-tunnel.ts`) und schützt explizit vor Argument Injection.
- Wenn der Agent jedoch `read`-Zugriff auf dem Host hat (kein Sandbox, oder Sandbox mit Bind auf `~/.ssh`), kann er Key-Dateien lesen. Sicherheit hängt von Tool-Policy + Sandboxing ab, nicht vom SSH-Codepfad selbst.

Browser-Credentials und Cookie-DBs

- Das Browser-Tool kann einen echten Browser steuern. Das Security-Dokument behandelt Browser-Profile als sensibel und empfiehlt ein dediziertes Profil, um Exposure zu vermeiden (`docs/gateway/security/i`
- OpenClaw enthält standardmäßig keinen Code, der Browser-Cookie-Datenbanken extrahiert. Es gibt ein Debug-Skript (`scripts/debug-claude-usage.ts`), das auf Firefox `cookies.sqlite` verweist, es ist aber nicht Teil der normalen Ausführung.
- Wenn `read` auf dem Host aktiv ist, kann ein Agent Browser-SQLite-DBs direkt lesen. Wenn das Browser-Tool auf ein eingeloggtes Profil zeigt, kann der Agent diese Sessions über die UI nutzen.

Fazit: Credentials sind nur dann "sicher", wenn Tool-Policy und Sandboxing so konfiguriert sind, dass Filesystem-Zugriff und/oder Browser-Control verhindert werden. Das System schützt dein Home-Verzeichnis nicht automatisch, wenn du diese Kontrollen nicht aktivierst.

Krypto-Wallet-Keys, E-Banking-Credentials, Keyrings

- OpenClaw hat keine nativen Integrationen für Krypto-Wallets oder Banking-Systeme. Es liest diese Secrets nicht, außer sie liegen in erreichbaren Dateien oder sind über Tools zugänglich.
- Keyring-Nutzung erscheint in Skills-Dokumentation (z. B. `skills/himalaya/SKILL.md` empfiehlt `pass` oder keyring). Diese Secrets liegen im OS-Keyring oder einem Password-Manager und werden nur gelesen, wenn der Agent den relevanten Command ausführen kann oder Zugang zum Keyring hat.

Praktisches Risiko: Wenn `exec` aktiviert ist und der OS-User ohne Prompt auf den Keyring zugreifen kann, kann der Agent Secrets wahrscheinlich abrufen. Ist `exec` deaktiviert (oder gesandboxed ohne Keyring-Zugriff), bleiben diese sicher.

Wie konfigurierbar ist OpenClaw? Kann ich Credentials oder Teile davon verstecken?

Ja. Wichtige Controls:

- **State-Verzeichnis isolieren**: OpenClaw-State via `OPENCLAW_STATE_DIR` in einen dedizierten Pfad verschieben, danach Rechte einschränken.
- **Sandboxing**: `agents.defaults.sandbox.mode` auf `"all"` setzen und `workspaceAccess` auf `"none"` oder `"ro"`, um den Zugriff aufs Home-Verzeichnis standardmäßig zu verhindern (`docs/gateway/sandboxing.md`).
- **Tool-Policy**: Hochrisiko-Tools (`read`, `exec`, `browser`, `process`) für untrusted Agents oder Channels verweigern (`docs/gateway/sandbox-vs-tool-policy-vs-elevated.md`).

- **Browser-Isolation**: dediziertes Profil nutzen; Browser-Control deaktivieren, wenn nicht benötigt (`docs/gateway/security/index.md`).
- **mDNS minimal mode**: Offenlegung von `cliPath`/`sshPort` via `discovery.mdns.mode: "minimal"` reduzieren (`docs/gateway/security/index.md`).
- **DM- und Gruppenrichtlinien**: Pairing/Allowlists reduzieren, wer Aktionen triggern darf (`docs/gateway/secu`

Du kannst "Teile" von Credentials verstecken, indem du env vars (`~/.openclaw/.env`) nutzt, State-Verzeichnisse pro Agent trennst und nur schmale Bind-Pfade in die Sandbox mountest. Das System unterstützt per-Agent Overrides für Sandbox und Tool-Policy, sodass "public" Agents ohne Filesystem-Zugriff parallel zu privaten Agents mit größerem Zugriff laufen können.

Wie steht es um Prompt Injection?

OpenClaw implementiert Prompt-Injection-Controls auf zwei Ebenen:

1. **Content Wrapping**: externe Inputs werden mit Warnungen und Boundary-Markern gekapselt, bevor sie an das Modell gehen (`src/security/external-content.ts`); Cron-Hooks wenden das standardmäßig an (`src/cron/isolated-agent/run.ts`).
2. **Operative Maßnahmen**: die Security-Dokumente empfehlen strikte inbound Access Controls, Mention-Gating in Gruppen, Sandboxing und Tool Allow/Deny als harte Grenzen.

Prompt Injection ist **nicht vollständig vermeidbar**. Das Design geht davon aus, dass das Modell manipulierbar ist und verlässt sich daher primär auf **Policy** + **Sandbox** + **Access Control**.

6) Praktische Hardening-Patterns

Konservative Profile, je nach Risikotoleranz:

A) Untrusted Agents stark einschränken (kein Filesystem/Exec/Browser)

```
{
  agents: {
    list: [
      {
        id: "public",
        sandbox: { mode: "all", scope: "agent", workspaceAccess: "none" },
        tools: {
          allow: ["group:sessions", "group:messaging"],
          deny: ["group:fs", "group:runtime", "browser", "process"]
        }
      }
    ]
  }
}
```

B) Read-only Analyse-Agent (sicherer File-Zugriff)

```
{
  agents: {
    list: [
      {
        id: "reader",
        sandbox: { mode: "all", scope: "agent", workspaceAccess: "ro" },
        tools: { allow: ["read"], deny: ["write", "edit", "apply_patch", "exec"] }
      }
    ]
  }
}
```

C) Separater OpenClaw-State

```
export OPENCLAW_STATE_DIR="/srv/openclaw-state"
chmod 700 /srv/openclaw-state
```

Das isoliert Credentials und Transkripte vom Home-Verzeichnis und macht die File-Grenze leichter zu schützen.

7) Wichtigste Takeaways

- **Kein Tool-Zugriff** = **starke Isolation**. Wenn du `read` und `exec` deaktivierst, verschwinden die meisten Credential-Exposition-Risiken.
- **Sandboxing ist kritisch**, wenn Tools aktiviert sind; es limitiert File- und Prozesszugriffe standardmäßig.
- **Browser-Control ist privilegiert**. Es gibt effektiven Zugriff auf alles, was das Browser-Profil erreichen kann.
- **Prompt Injection ist gemindert, aber nicht "gelöst."** Operative Controls (Allowlists, Sandboxing, Tool-Policy) sind das eigentliche Sicherheitsnetz.

Bericht generiert von GPT-5.2-Codex, gepromptet am 5. Februar 2026 von GRG in openclaw/openclaw:main