

cs-console

Handbuch

Flüchtiges interaktives PTY-Relay pro Anfrage mit Betriebssystem-Passwortsperre für System > Console und den AI-Helpdesk

Version 0.5.8 · 13. Oktober 2026

Inhalt

1. Überblick	3
2. Unterstützte Plattformen	3
3. Passwortsperr	3
4. Expect-Modus	3
5. Verhalten und Optionen	4
6. Einsatz in napp-it CS	4
7. Lizenz	4

1. Überblick

cs-console ist das flüchtige, pro Anfrage gestartete interaktive PTY-Relay von napp-it 4ai (Client-Server). Es wird von `server.pl` bei Bedarf für **System > Console** und für den Konsolenmodus des AI-Helpdesks gestartet, öffnet genau eine direkte Verbindung, relayt eine interaktive Shell und beendet sich dann.

Sicherheit durch Konstruktion:

- es verlangt immer einen root-/Administrator-Elternprozess;
- im interaktiven Shell-Modus verlangt es zusätzlich eine **Passwortbestätigung über die Konsole selbst** (die Passwortsperr, Kapitel 3);
- die Verbindung ist per IP und Token beschränkt und mit ChaCha20-Poly1305 versiegelt.

Der Entwurf folgt csweb-gui/data/howto.ai/cs-console.info. csweb-gui verteilt und aktualisiert das Binary pro Member über das Menü **About > Download cs-tools**.

Dies ist ein Release Candidate. v0.5.5 liegt auf GitHub mit fertigen Binaries für 8 Ziele (windows, linux, darwin, freebsd, illumos, solaris amd64; linux und darwin auch arm64). Die interaktive Konsole (Öffnen, Betriebssystem-Passwortsperr, Shell) wurde auf allen sechs amd64-Plattformen durchgehend live verifiziert: Windows, Linux (Proxmox), illumos (OmniOS), macOS, FreeBSD und Solaris.

2. Unterstützte Plattformen

Plattform	Architektur	PTY-Backend	Status
Linux	amd64, arm64	creack/pty	live verifiziert
Windows	amd64	ConPTY	live verifiziert
macOS	amd64, arm64	creack/pty	live verifiziert
FreeBSD	amd64	creack/pty	live verifiziert
illumos	amd64	selbstgebautes STREAMS (cgo)	live verifiziert
Solaris	amd64	creack/pty	live verifiziert

Der Transport ist POLL (nicht SSE) über den Perl-Webserver, mit einer eigenen, ratenlimit-freien Authentifizierung für die Endpunkte `/console/*`.

3. Passwortsperr

Bevor das PTY startet, bestätigt das Betriebssystem selbst das root-/Administrator-Passwort: PAM unter Linux, macOS, FreeBSD, illumos und Solaris, `LogonUser` unter Windows. Es gibt bis zu 3 Versuche und eine Sperr je IP von 15 s (lockout.go). Das Passwort - oder ein von PAM verlangtes OTP - erreicht nie das KI-Modell, das Chat-Protokoll oder die Logs. Die Passwortsperr reicht den tatsächlichen PAM-/LogonUser-Fehlertext durch (nur Modul- und Fehlernamen, nie Geheimnisse) statt eines bloßen "authentication failed".

Parallele Sitzungen von derselben Frontend-IP teilen sich ein Versuchskontingent: Sie bekommen nicht mehr je ein eigenes 3-Versuche-Kontingent (die Nebenläufigkeits-Korrektur von v0.5.7, abgedeckt durch `lockout_concurrency_test.go`, `-race` sauber).

4. Expect-Modus

Neben der interaktiven Shell gibt es freigegebene Expect-Aktionen. Sie steuern ein interaktives Programm mit vorbereiteten Eingabeaufforderungen, sodass ein Geheimnis einmal in das Programm getippt und nicht auf einer Kommandozeile übergeben wird.

- `passwd_user`, `smbpasswd_user`, `ksmbd_user` - das Passwort eines Benutzers setzen (root / Administrator, uid 0, wird immer abgelehnt).

- `zfs_create_enc_prompt` und `zfs_unlock_enc_prompt` (v0.5.8) - ein verschlüsseltes Dataset mit `keylocation=prompt` anlegen oder entsperren, indem die Passphrase interaktiv getippt wird. Das ist nötig, weil Solaris 11 und Windows (OpenZFS on Windows) beide nicht-interaktives stdin für die ZFS-Schlüsseingabe im Prompt-Modus ablehnen - eine interaktive Eingabe ist auf diesen beiden Plattformen der einzige Weg. Das dateibasierte Anlegen von napp-it und keysplit bleiben unverändert; sie brauchen echte Schlüssel-Bytes auf der Platte, die eine Eingabeaufforderung nicht liefern kann. Alle vier Prompt-Texte (Solaris anlegen/entsperren, Windows anlegen/entsperren) wurden vor der Auslieferung live erfasst und geprüft.

Robustheits-Korrekturen für jede Expect-Aktion: das ConPTY-Zeilenende unter Windows (ein einzelnes `\n` schickte dort nie eine Zeile ab; jetzt `\r\n` über ein GOOS-abhängiges `lineEnding()`), ANSI-Escape-Folgen mitten in einer Eingabeaufforderung (`norm()` entfernt sie vor dem Abgleich) und die Anlege-Eingabeaufforderung von illumos, die sich von Windows unterscheidet ("Enter passphrase:" / "Re-enter passphrase:") und jetzt getrennt für Solaris, Windows und den Standard aufgelöst wird.

5. Verhalten und Optionen

- `main.go` liest die Startkonfiguration von stdin, startet das PTY, nimmt eine direkte Verbindung mit IP- und Token-Authentifizierung an, relayt mit ChaCha20-Poly1305 und beendet sich dann.
- `NO_COLOR` wird nicht mehr erzwungen: Das dunkle xterm.js-Thema stellt ANSI-Farben problemlos dar. Reiner Text lässt sich mit `startConfig.no_color` oder `CS_CONSOLE_FORCE_NO_COLOR=1` wieder wählen.
- Eine Verbindung durch den falschen Gegenüber wird mit einem erkennbaren Sentinel-Fehler gemeldet, sodass ein Aufrufer ihn in eine verständliche Meldung übersetzen kann.

Quellcode-Aufbau: `main.go` (Startkonfiguration, PTY-Start, Relay), `pty.go` mit `pty_unix.go`, `pty_windows.go` und `pty_illumos.go` (PTY-Backends), `session.go` und `crypto.go` (Eltern-Protokoll und versiegeltes Relay), `auth.go` mit `auth_unix.go` und `auth_windows.go` (Passwortsperre), `lockout.go` (Brute-Force-Sperre je IP) und `expect.go` (freigegebene Expect-Aktionen).

6. Einsatz in napp-it CS

- **System > Console** öffnet im Browser eine Konsole des Members. Das Binary wird pro Anfrage gestartet und endet mit der Sitzung.
- Der **AI-Helpdesk** kann für Fehlersuche und Setups eine interaktive Root-Konsole steuern - nur nach einer Bestätigung und einer manuellen Passworteingabe über die Konsole selbst.
- Das Binary liegt in `data/cs_server/tools/cs-console/<os>.<arch>/` und wird über **About > Download cs-tools** verteilt; es wird nicht automatisch heruntergeladen.
- Noch offene Verdrahtung: die Menü-Aktion der Web-GUI für die neuen Prompt-Aktionen und die Spiegelung der Freigabeliste in `server.pl` und `aihelplib`.

7. Lizenz

BSD 2-Clause - siehe Datei LICENSE.