

Knowledge Base

Aktive technische Knowledge-Base-Einträge aus Obsidian.

- [RustDesk in Tactical RMM Integrieren](#)
- [TRMM-Linux-Agent-Konzept](#)

RustDesk in Tactical RMM Integrieren

Wiki: RustDesk in Tactical RMM Integrieren

“ **note:** > Aktueller CORESTREAM-Runbook-Entwurf: RustDesk Self-hosted Client Setup

Quellen:

<https://rustdesk.com/docs/de/self-host/client-deployment/integrations/tactical-rmm/>

<https://github.com/matlat8/TRMM-RustDesk-Integration>

Installationskript RustDesk

Mit diesem Skript wird der RustDesk Client installiert.

```
$ErrorActionPreference= 'silentlycontinue'

if (!(Test-Path C:\Temp))
{
    New-Item -ItemType Directory -Force -Path C:\Temp > null
}

cd C:\Temp

Invoke-WebRequest "https://github.com/rustdesk/rustdesk/releases/download/1.2.6/rustdesk-1.2.6-x86_64.exe" -Outfile "rustdesk.exe"
Start-Process .\rustdesk.exe --silent-install -wait

$ServiceName = 'Rustdesk'
```

```

$arrService = Get-Service -Name $ServiceName -ErrorAction SilentlyContinue

if ($arrService -eq $null)
{
    Start-Sleep -seconds 20
}

while ($arrService.Status -ne 'Running')
{
    Start-Service $ServiceName
    Start-Sleep -seconds 5
    $arrService.Refresh()
}

net stop rustdesk

$username = ((Get-WMIObject -ClassName Win32_ComputerSystem).Username).Split('\')[1]
Remove-Item C:\Users\$username\AppData\Roaming\RustDesk\config\RustDesk2.toml
New-Item C:\Users\$username\AppData\Roaming\RustDesk\config\RustDesk2.toml
Set-Content C:\Users\$username\AppData\Roaming\RustDesk\config\RustDesk2.toml
"rendezvous_server = 'IPADDRESS' `n nat_type = 1 `n serial = 0 `n `n [options] `n custom-rendezvous-
server = 'IPADDRESS' `n key = 'KEY=' `n relay-server = 'IPADDRESS' `n api-server =
'https://remote.corestream.ch'"
Remove-Item
C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\RustDesk\config\RustDesk2.toml
New-Item
C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\RustDesk\config\RustDesk2.toml
Set-Content
C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\RustDesk\config\RustDesk2.toml
"rendezvous_server = 'IPADDRESS' `n nat_type = 1 `n serial = 0 `n `n [options] `n custom-rendezvous-
server = 'IPADDRESS' `n key = 'KEY=' `n relay-server = 'IPADDRESS' `n api-server =
'https://remote.corestream.ch'"

net start rustdesk

```

TRMM-Linux-Agent-Konzept

“ **abstract:** Kurzfassung Linux-Hosts in TRMM laufen bei CORESTREAM inzwischen über einen **eigenen Linux-Agent-Fork** mit vorgebauten, SHA256-verifizierten Linux-Binaries und mobile-freundlichem `curl | bash`-Installer. Der frühere netvolt-/Live-Compile-Workaround entfällt. Der Pilot mit `v2.11.0-cs4` wurde erfolgreich getestet; Mesh + `tacticalagent.service` funktionieren.

1. Problemstellung

“ **warning:** Show-Stopper im MSP-Alltag Linux-Hosts lassen sich in TRMM nur über einen **umständlichen, nicht automatisierbaren Workaround** aufnehmen. Pro Maschine von Hand — nicht skalierbar.

- **Ursache = TRMM Paywall:** Signierte Agenten (inkl. Linux) liefert TRMM nur an zahlende Sponsoren (Tier 1, monatlich) über einen eigenen Code-Signing-Server. Agent-Generate ohne Token → **HTTP 400** → kein offizieller Linux-Agent.
- **Workaround = netvolt-Script:** pro Maschine manuell ausgeführt (Go live installieren, `master` live kompilieren, systemd einrichten). Sicherheitstechnisch bedenklich: **ungepinntes** `master`, Compile auf Kunden-Server.
- **Fakt (geprüft im Code):** Die 400 ist eine **Paywall, kein Linux-Technik-Erfordernis**. `AgentUpdate` (`agent_unix.go`) prüft **keine** Signatur — auf Linux ist Code-Signing irrelevant.

“ **success:** Lösung Eigenen Fork + Build-CI betreiben → vorgebaute, SHA256-verifizierte Linux-Binaries + sauberes `curl | bash`-Install-Script. **Kein TRMM-Token, keine Paywall, kein manueller Umweg.**

2. Ausgangslage (faktisch)

- Offizieller Agent **kann Linux:** `agent_unix.go` + `install_unix.go`, baut sauber (`CGO_ENABLED=0 GOOS=linux ... go build -ldflags "-s -w"`).

- TRMM vertreibt Linux-Binary **offiziell nicht** → daher der netvolt-Umweg.
- `netvolt/LinuxRMM-Script` macht 3 Dinge:
 1. Go auf dem Zielhost installieren (`go_install`)
 2. Agent aus `amidaware/rmmagent` **master** live kompilieren (`agent_compile`)
 3. systemd-Unit + Mesh-Installer drüberlegen

“ **bug:** Sicherheitsproblem des Umwegs Lädt Go vom Web, clont `master` (ungepinnt!) und kompiliert **zur Laufzeit auf dem Kunden-Server**. Exakt die „fremdes Binary/Source vertrauen“-Falle aus dem 2021/22 Miner-Vorfall — nur als Live-Source statt als Binary.

3. Zielbild

Eigener Fork mit **vorgebauten, verifizierten** Linux-Binaries + sauberem Installer.

- ☒ Kein Go auf dem Client, kein Live-Compile
- ☒ Source-Commit + Binary per SHA256 verifiziert
- ☒ Update-Pfad zeigt auf das eigene Repo

4. Komponenten

4.1 Fork-Strategie

- Repo: **internes Git** (`git.corestream.ch`, OneDev) statt öffentliches GitHub-Repo
- Fork/Mirror von `amidaware/rmmagent` als internes Projekt, z.B. `corestream/rmmagent`
- Branches: `main` (= CORESTREAM) + `upstream` als Remote/Mirror
- Upstream-Commits **nur geprüft** mergen (Review-Diff + `go mod verify`); Sicherheitsfixes priorisieren

4.2 Build-CI (erzeugt die Binaries)

- Laufzeit: **interne Build-VM / internes CI** statt GitHub Actions
- Matrix: `linux/amd64`, `linux/arm64`, `linux/386`
- Build: `CGO_ENABLED=0 GOOS=linux GOARCH=<arch> go build -ldflags "-s -w" -o rmmagent-linux-<arch>`
- Artefakte: `rmmagent-linux-{arch}`, `SHA256SUMS`, optional zusätzlich mit internem Schlüssel signiert

- Storage: **internes Git/Release- oder Artefakt-Repo** (gepinnte Tags, z.B. `v2.8.0-cs1`), nicht öffentlich

4.3 Install-Script (Ersatz für netvolt)

- Öffentlicher Installer: `https://git.corestream.ch/corestream/rmmagent/~raw/main/install-linux.sh`
- Standard-One-Liner: `curl -fsSL https://git.corestream.ch/corestream/rmmagent/~raw/main/install-linux.sh | sudo bash -s -- -c CLIENT -s SITE -t TOKEN`
- Wenn `-c`, `-s` oder `-t` fehlen, fragt das Script interaktiv via `/dev/tty` nach; Token/Auth wird verdeckt gelesen.
- Lädt **vorgebautes** Binary vom internen Fork-Repo (aktuell `v2.11.0-cs4`, nicht `master`).
- **SHA256-Verifikation vor Install** gegen `SHA256SUMS`.
- Installiert Mesh standardmässig direkt via MeshCentral Linux-Agent-URLs, weil `/api/v3/meshexe/` Linux nicht zuverlässig liefert.
- Liest nach der Mesh-Installation die lokale Mesh Node ID aus und übergibt sie an `/api/v3/newagent/`.
- Legt systemd-Unit `tacticalagent.service` an (`Restart=always`) und startet/aktiviert sie.
- **Kein** `go_install`, **kein** `agent_compile` auf dem Client.

4.4 Update-Pfad

- Linux `AgentUpdate` ist im Fork gehärtet: Update-Binary wird nur ersetzt, wenn die SHA256 gegen ein benachbartes `SHA256SUMS` passt.
- Server-/Update-URLs müssen auf das interne Artefaktlayout zeigen: `<BASE_URL>/<VERSION>/rmmagent-linux-<arch>` plus `<BASE_URL>/<VERSION>/SHA256SUMS`.

4.5 Security (aus Miner-Review)

- Reproducible Builds, Checksum-Verifikation im Installer + Agent-Update
- Gepinnter Source-Tag (kein `master`)
- Build-Pipeline selbst reviewen (realistischere Angriffsfläche als Source)
- `go mod verify` im CI
- Niemals fremde vorgefertigte Binaries an Kunden ausliefern

5. Linux-Code Gaps

Funktionen in `agent_unix.go` sind oft **Stub/nil** — Linux kann weniger als Windows:

Funktion	Linux-Status	Bedeutung
GetDisks / checkin / RunScript / Terminal / Mesh	❑ implementiert	Kern-RMM funktioniert

Funktion	Linux-Status	Bedeutung
GetInstalledSoftware	<code>{}</code> (leer)	Keine Software-Inventur
SendSoftware	<code>{}</code>	kein Reporting
GetServiceDetail / ControlService / EditService	Windows-only	Service-Steuerung nur via Script
PatchMgmt	<code>nil</code>	kein Patch-Management
CreateSchedTask / ListSchedTasks	<code>nil</code>	kein GUI-Task-Scheduling
GetWinUpdates / InstallUpdates	Windows	kein OS-Update-Mgmt
GetEventLog	Windows	kein Eventlog
InstallChoco	Windows	kein Chocolatey

“ **note:** Fazit Linux-Agent taugt für **Monitoring, Scripts, Terminal, Mesh** — aber keine Software-Inventur, kein Patch-Mgmt, keine Service-UI. Feature-Gap, kein Bug. *Entscheidung: Gaps füllen oder „Monitoring-only“ akzeptieren?*

6. Code-Signing-Strategie (Linux-fokussiert)

Plattform	Signing nötig?	Maßnahme
Linux	☐ nein	SHA256-Verifikation statt Signatur (reicht, da wir Build kontrollieren)
macOS	☐ vorerst NEIN (2026-07-28)	kein Apple-Dev-Account; bei Bedarf später nachrüsten
Windows	☐ vorerst raus	läuft weiter via TRMM-Sponsor / unsigned + AV-Exclusion

“ **info:** Kernpunkt Der 400-Fehler verschwindet komplett, sobald wir die Build-CI **selbst** betreiben — kein TRMM-Token, keine Paywall, kein manueller netvolt-Umweg mehr.

7. Risiken

caution: Wichtig

- **Fork-Pflegeverantwortung** dauerhaft (Build-Kette + Security-Tracking)
- **CVEs** manuell verfolgen (kein upstream-Auto-Patch)
- **Lizenz:** Tactical RMM License v1.0 ist **kein** OSI-Open-Source — verbietet kommerziellen Weiterverkauf. Interne MSP-Nutzung OK;
Rebrand nur intern, kein gebrandeter SaaS-Verkauf

8. Rollout (Phasen)

1. **Dev-VM:** Fork + CI + Install-Script testen (Linux-VM bei euch)
2. **Pilot:** 1-2 eigene CORESTREAM-Hosts
3. **Prod:** ausgewählte Kunden-Linux-Server

8b. Scaffold-Stand 2026-08-17

“ **success:** Erster interner Scaffold gebaut Upstream `amidaware/rmmagent` wurde auf Tag `v2.11.0` / Commit `e6b61ba540af2df9d13247f96538397daa3c4f1f` gepinnt, statisch geprüft und lokal als CORESTREAM-Scaffold vorbereitet.

Lokaler Arbeitsstand auf Hermes:

- Repo: `/home/csgadmin/work/trmm-linux-agent/corestream-rmmagent`
- Branch: `main`
- Commit `df2b116`: Import upstream `v2.11.0`
- Commit `252e601`: CORESTREAM Build-/Installer-Scaffold
- Commit `e54fa3f`: Installer-One-Liner-Workflow verbessert (`--api-url/--client-id/--site-id/--auth/--base-url`), Preflight, Dry-run, klare Fehlerausgaben, One-Liner-Generator)
- Interner Remote vorbereitet: `https://git.corestream.ch/corestream/rmmagent.git`
- Push noch offen: OneDev verlangt HTTPS-Credentials/Token

Gebaut/verifiziert:

- Dockerized Go `go1.24.13` verwendet — kein Go lokal auf Hermes installiert
- `go mod verify`: erfolgreich
- Linux-Builds: `amd64`, `arm64`, `386` erfolgreich

- Self-Update-Härtung: `AgentUpdate` lädt künftig ein benachbartes `SHA256SUMS` und ersetzt die Binary nur bei passender SHA256
- Unit-Tests für positive und negative Checksum-Verifikation: erfolgreich
- Installer-Wrapper: `scripts/install-linux-corestream.sh` — jetzt one-liner-fähig mit CLI-Flags, Preflight, klaren `[OK]` / `[FEHLER]`-Ausgaben und `--dry-run` für Download/SHA256-Test ohne Installation
- One-Liner-Generator: `scripts/render-linux-oneliner.sh` erzeugt einen sauber gequoteten kunden-/site-spezifischen Installationsbefehl
- Build-Script: `scripts/build-linux.sh`
- Audit-Report: `docs/audit-v2.11.0.md`

Artefakte lokal:

- `/home/csgadmin/work/trmm-linux-agent/corestream-rmmagent/dist/v2.11.0-cs2/`
- `/home/csgadmin/work/trmm-linux-agent/packages/corestream-rmmagent-scaffold-v2.11.0-cs3.tar.gz`
- `/home/csgadmin/work/trmm-linux-agent/packages/corestream-rmmagent-scaffold-v2.11.0-cs2.tar.gz`
- `/home/csgadmin/work/trmm-linux-agent/packages/corestream-rmmagent-linux-artifacts-v2.11.0-cs2.tar.gz`

Checksums der Linux-Binaries:

- `rmmagent-linux-amd64`: `4215c699fb5526a1b3567cf91cff6011c300a2634a87d8755e301eedbf471925`
- `rmmagent-linux-arm64`: `752bb9b9247c9e1e4e1e479c752a53f9ef9d759f6474f7853a9b33bbca055899`
- `rmmagent-linux-386`: `9ad05849fe0fdd470a46c4b52611113f81f46a613f61955236965ac1ff5176e0`

Package-Checksums:

- `corestream-rmmagent-scaffold-v2.11.0-cs3.tar.gz`: `9b25c2e476f8c80b06a4bb11212e1796bc294fcaa5ed944e05fa80510d2bb3ef`
- `corestream-rmmagent-scaffold-v2.11.0-cs2.tar.gz`: `372fda6e73d5c50b2938ca073550b35a42568977aeec531fc2cd013d5ab8f5ec`
- `corestream-rmmagent-linux-artifacts-v2.11.0-cs2.tar.gz`: `2991b7560ec2492b17ba1ad8d353ea05af3070a2373b16e3fdafa4aafba8fcb1`

“ **success:** Self-Update-Härtung erledigt Der Installer und der Linux-Self-Update-Pfad verifizieren jetzt SHA256. Voraussetzung: Update-URLs zeigen auf internes Artefaktlayout `<BASE_URL>/<VERSION>/rmmagent-linux-<arch>` und daneben liegt `SHA256SUMS` .

8c. Produktiver Installer-Stand 2026-09-09

success: Pilot bestätigt Der kurze Linux-Installer wurde mit `v2.11.0-cs4` auf einer Testmaschine erfolgreich ausgeführt. Agent registriert sich im RMM, Mesh wird installiert und `tacticalagent.service` läuft.

Aktueller Prozess für Techniker:

1. In Tactical RMM Client/Site und Enrollment-/Deployment-Token bestimmen.
2. Auf Linux-Zielhost folgenden Befehl ausführen:

```
curl -fsSL https://git.corestream.ch/corestream/rmmagent/~raw/main/install-linux.sh |  
sudo bash -s -- -c CLIENT -s SITE -t TOKEN
```
3. Wenn Werte nicht direkt im Befehl stehen, fragt der Installer interaktiv nach:
 - `Client-ID`
 - `Site-ID`
 - `Enrollment-Token/Auth` verdeckt
4. Nach Installation prüfen:
 - RMM: Host erscheint unter der gewählten Client/Site.
 - Systemd: `tacticalagent.service` läuft.
 - Mesh: Remote-Control-Verknüpfung ist vorhanden.

Aktueller Artefaktstand `v2.11.0-cs4` im OneDev-Raw-Pfad:

- `rmmagent-linux-386`: `52e81126768a95f64cfc52fcf4c4641c7d6a95b09db09e1d83804c14378d1843`
- `rmmagent-linux-amd64`: `480aa5d95b33a6cc6bf025c5b850cb1ae989a2e7cfea42c97d114c2559420900`
- `rmmagent-linux-arm64`: `eea2d66389a83a7cd7193599310b96d5722deafd664beff3e553e720f227702f`

Relevante Commits:

- `28ee3a2` — Installer legt `tacticalagent.service` selbst an.
- `ca80646` — Linux Mesh Node ID wird während Installation synchronisiert.
- `f11af1a` — Linux Mesh-Agent wird direkt von MeshCentral geladen.

Offen für nächsten Ausbau:

- Saubere Deployment-URL im RMM-Backend, z.B.
`https://api.corestream.ch/clients/<uid>/deploy/linux.sh` oder Proxy-Alias
`https://rmm.corestream.ch/i/<uid>`, damit Techniker keinen Token mehr manuell kopieren müssen.

9. Offene Entscheidungen

- ☒ CI: **interne Build-VM / internes CI** statt GitHub Actions
- ☒ Artefakt-Storage: **internes Git/Release- oder Artefakt-Repo**

- ☐ Rebrand-Umfang (Name/Logo) — Lizenzgrenze beachten
 - ☐ Linux-Gaps füllen oder als „Monitoring-only“ akzeptieren?
 - ☐ Update-Pfad: voll eigen oder upstream-Releases spiegeln?
 - ☒ **macOS: vorerst NEIN** (2026-07-28) — kein Apple-Dev-Account nötig
-

Verwandt

- [Knowledge Base/2026/Linux/](#) — weitere Linux-Themen
 - [CORESTREAM/Konzepte/](#) — CORESTREAM-Architektur
-

Quelle: Obsidian [Knowledge Base/TacticalRMM/TRMM-Linux-Agent-Konzept.md](#) · Stand: 2026-09-20 ·
Automatisiert publiziert durch Hermes.