Tactical RMM

Tactical RMM ist ein Open-Scource Tool zur Fernüberwachung und -verwaltung, das mit Django, Vue und Golang entwickelt wurde. Es verwendet einen in Golang geschriebenen Agenten und ist mit MeshCentral integriert.

- Aktualisieren des RMM
- Aktualisieren des Let's Encrypt SSL-Zertifikat
- Linux Agent Installation & Management
- Knowledge Base
    - RustDesk in Tactical RMM integrieren
    - TRMM-Linux-Agent-Konzept

Aktualisieren des RMM

Server auf dem neuesten Stand halten

Sie sollten in regelmäßigen Abständen `sudo apt update` und `sudo apt -y` upgrade ausführen, um Ihren Server auf dem neuesten Stand zu halten. Ansonsten sollten Sie keine Änderungen an Ihrem Server vornehmen und alles andere dem Skript update.sh überlassen.

> Versuchen Sie NICHT, Betriebssystem-Upgrades an Ort und Stelle durchzuführen, wie z.B. ein Upgrade von Debian 11 auf 12 an Ort und Stelle. Sie werden Ihren Server zerstören!

Aktualisierung auf die neueste RMM-Version¶

> Machen Sie vor dem Update immer ein Backup rsp. ein Snapshot der aktuellen Installation

Melden Sie sich per SSH an Ihren Server an mit dem Benutzer, den Sie während der Installation angelegt haben.

> Führen Sie niemals Update-Skripte oder -Befehle als Root-Benutzer aus. Das bringt die Berechtigungen durcheinander und macht Ihre Installation kaputt.

Laden Sie das Update-Skript herunter und führen Sie es aus:

```
wget -N https://raw.githubusercontent.com/amidaware/tacticalrmm/master/update.sh
chmod +x update.sh
./update.sh
```

Wenn Sie bereits auf der neuesten Version sind, wird das Update-Skript Sie darüber informieren und sofort zurückkehren.

Sie können die optionale Option --force an das Update-Skript übergeben, um ein Update zu erzwingen, wodurch die Prüfung auf die neueste Version umgangen wird.

```
wget -N https://raw.githubusercontent.com/amidaware/tacticalrmm/master/update.sh
chmod +x update.sh
./update.sh --force
```

Dies ist nützlich für eine verpfuschte Aktualisierung, die möglicherweise nicht vollständig abgeschlossen wurde.

Das Update-Skript korrigiert auch alle Berechtigungen, die während eines verpfuschten Updates durcheinander geraten sind, oder wenn Sie das Update-Skript versehentlich als Root-Benutzer ausgeführt haben.

> Versuchen Sie nicht, MeshCentral manuell auf eine neuere Version zu aktualisieren. Sie sollten dies dem update.sh-Skript überlassen. Die Entwickler testen MeshCentral und stellen sicher, dass die Integration nicht unterbrochen wird, bevor sie die Mesh-Version aktualisieren.

Aktualisieren des Let's Encrypt SSL-Zertifikat

> Derzeit erneuert das Update-Skript Ihr Let's Encrypt-Wildcard-Zertifikat, das alle 3 Monate abläuft, nicht automatisch, da es nicht ohne weiteres möglich ist, dies mit der DNS-TXT-Eintragsmethode zu automatisieren.

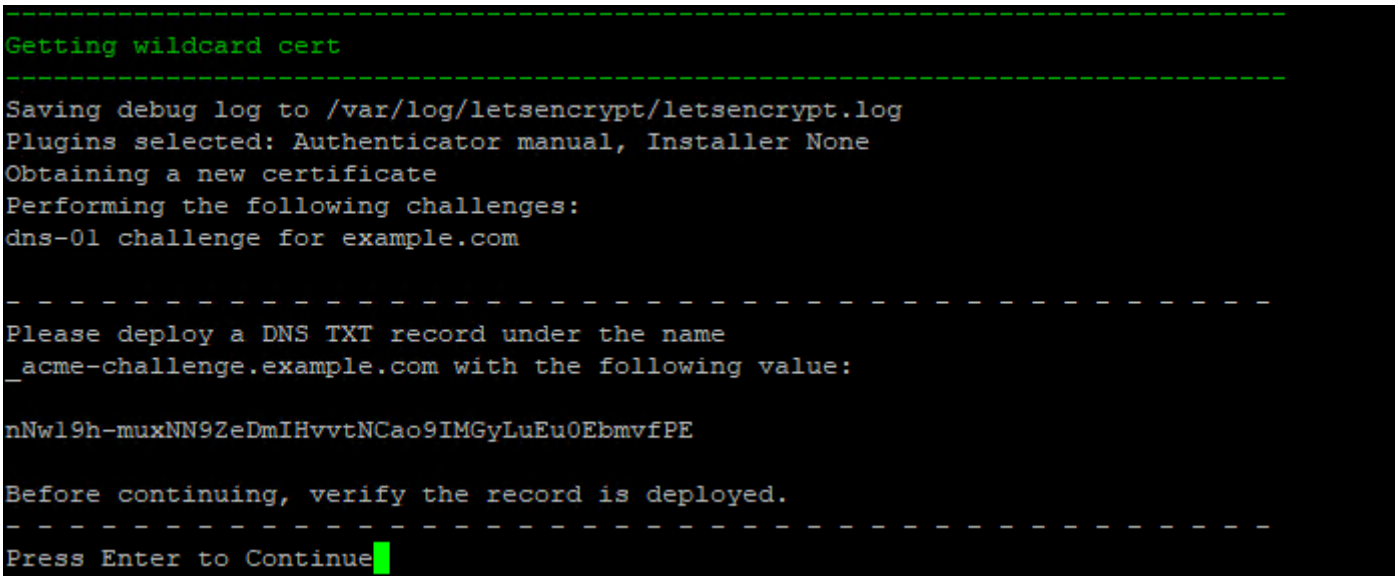
Let's Encrypt Wildcard Zertifikat erneuern

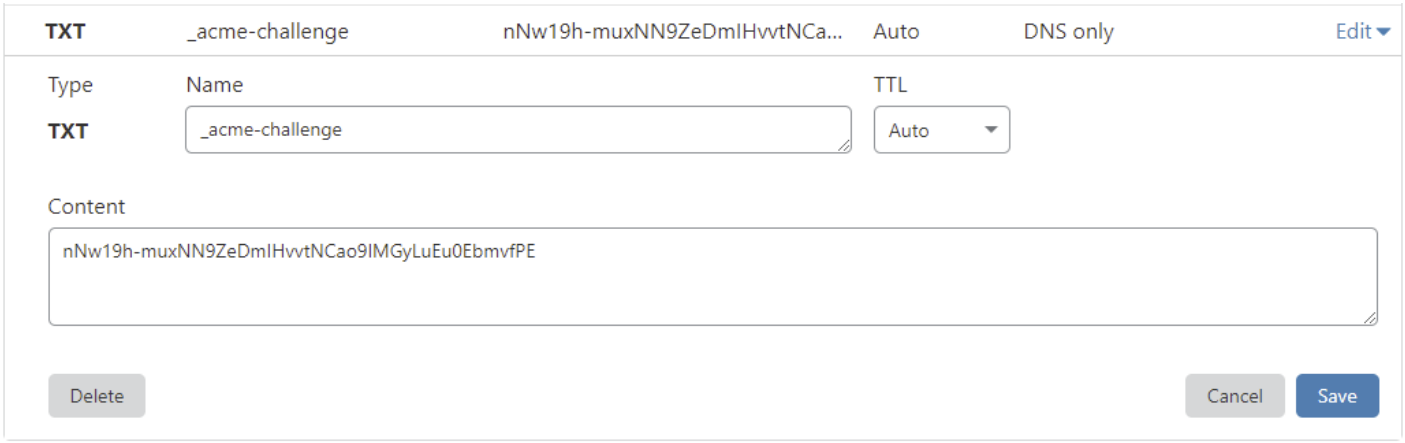
Um Ihre Let's Encrypt Wildcard Cert SSL-Zertifikate zu erneuern und zu aktualisieren, führen Sie den folgenden Befehl aus, wobei Sie `example.com` durch Ihre Domain und `admin@example.com` durch Ihre E-Mail-Adresse ersetzen:

```
sudo certbot certonly --manual -d *.example.com --agree-tos --no-bootstrap --preferred-challenges dns -m admin@example.com --no-eff-email
```

DNS TXT Eintrag aktualisieren

> Die Propagierung von TXT-Einträgen kann je nach DNS-Anbieter zwischen 1 Minute und einigen Stunden dauern.
> Bevor Sie die Eingabetaste drücken, sollten Sie überprüfen, ob der TXT-Datensatz bereitgestellt wurde.
> Mit dem folgenden Befehl können Sie dies schnell überprüfen:
> `dig -t txt _acme-challenge.example.com` (nicht vom TRMM-Server)
> oder testen Sie mit: https://viewdns.info/dnsrecord/ Enter: `_acme-herausforderung.beispiel.com`





Neues Wildcard Zertifikat einbinden

Das erneuerte Zertifikat wird immer noch vollständig in Tactical verwendet. Führen Sie nun das Update Skript.

```
./update.sh --force
```

Linux Agent Installation & Management

□□ Zweck

Dieses Dokument beschreibt die Installation, Aktualisierung und Deinstallation des Linux-Agents über das bereitgestellte `linuxrmm.sh`-Skript.
Das Skript bindet Linux-Systeme an TacticalRMM / MeshCentral an und automatisiert die gesamte Einrichtung.

□□ Voraussetzungen

- Root- oder sudo-Zugriff auf das Zielsystem
- Unterstützte Distributionen: **Debian / Ubuntu** (getestet)
- Internetzugang zum TacticalRMM- und MeshCentral-Server
- Abhängigkeiten: `curl`, `wget`, `tar`, `systemd`

□□ Parameterquellen in TacticalRMM

Da die Funktion **„Add Linux Agent“** in unserer Umgebung derzeit **nicht aktiv** ist, müssen die Parameter **manuell** eingetragen werden.
Die folgenden Werte können direkt im TacticalRMM-Dashboard abgelesen werden:

Parameter	Beschreibung	Wo im TacticalRMM zu finden
`<MESH_URL>`	Download-Link des Mesh Agents (Linux Installer)	TacticalRMM → *Settings* → *Global Settings* → *MeshCentral* → Abschnitt **Mesh Settings** → Eintrag **"Linux Agent Download URL"**

Parameter	Beschreibung	Wo im TacticalRMM zu finden
`<API_URL>`	URL zur TacticalRMM API	TacticalRMM → *Settings* → *Global Settings* → Feld **"Base API URL"** (z. B. `https://rmm.example.com`)
`<CLIENT_ID>`	Eindeutige Kunden-ID	TacticalRMM → *Clients* → Kunde auswählen → Browser-Adresszeile (`/clients/3`) oder Tooltip „Client ID“
`<SITE_ID>`	Standort-ID des Kunden	TacticalRMM → *Clients* → *Sites* → Site auswählen → ID in der Adresszeile (`/sites/1`)
`<AUTH_KEY>`	Installationsschlüssel (Authentifizierung)	TacticalRMM → *Settings* → *API Keys* → Kategorie **Agent Auth Keys** → Neuen Key erstellen oder bestehenden verwenden
`<AGENT_TYPE>`	Typ der Installation (`server` / `workstation`)	Nach internem Standard definieren – dient der Unterscheidung im Dashboard

> ⚠ **Hinweis:**
> Auth-Keys und API-URLs sind **sensibel** und dürfen **nicht** in öffentlichen oder gemeinsam genutzten Dokumenten gespeichert werden.
> Alle Parameter können auch über Administrator-Zugriff direkt aus den TacticalRMM-Einstellungen exportiert werden.

⚙ Installation

□□ Syntax

```
sudo bash linuxrmm.sh install <MESH_URL> <API_URL> <CLIENT_ID> <SITE_ID> <AUTH_KEY>
<AGENT_TYPE>
```

□□ Beispiel

```
sudo bash linuxrmm.sh install \
"https://mesh.example.com/agent-linux64.sh" \
"https://rmm.example.com" \
```

```
3 1 "abcdefgh123456789" "server"
```

Das Skript lädt den passenden Agent (basierend auf der Systemarchitektur) herunter, konfiguriert ihn automatisch für den Mandanten und startet den Systemdienst.

□□ Update des Linux-Agents

```
sudo bash linuxrmm.sh update
```

Der laufende Dienst wird gestoppt, aktualisiert und automatisch neu gestartet.
Verbindungen zu TacticalRMM und MeshCentral bleiben erhalten.

□ Deinstallation

□□ Syntax

```
sudo bash linuxrmm.sh uninstall <MESH_FQDN> <MESH_ID>
```

□□ Beispiel

```
sudo bash linuxrmm.sh uninstall mesh.example.com A1B2C3D4E5
```

Der lokale Dienst und alle Dateien werden entfernt.
Der Eintrag im TacticalRMM-Dashboard bleibt bestehen und kann dort manuell gelöscht werden.

□□ Status- und Fehlerdiagnose

Befehl	Beschreibung
`sudo systemctl status linux-agent`	Zeigt den aktuellen Status des Agent-Dienstes
`journalctl -u linux-agent -n 50`	Zeigt die letzten 50 Log-Zeilen
`cat /var/log/linuxrmm/install.log`	Zeigt detaillierte Installations- und Laufzeit-Logs

Befehl	Beschreibung
`sudo bash linuxrmm.sh update`	Erzwingt eine manuelle Aktualisierung

□□ Ordnerstruktur

Pfad	Inhalt
`/opt/linuxrmm/`	Hauptverzeichnis des Agents
`/opt/linuxrmm/conf/`	Konfiguration, Client- und Site-Informationen
`/opt/linuxrmm/bin/`	Agent-Binary und Startskripte
`/var/log/linuxrmm/`	Installations- und Laufzeit-Logs

⚠ Häufige Probleme

Fehler	Ursache / Lösung
`unsupported arch`	Architektur nicht unterstützt → `uname -m` prüfen
Dienst startet nicht automatisch	`sudo systemctl enable --now linux-agent` ausführen
Keine Verbindung zum Dashboard	Firewall / Proxy prüfen
API-Auth-Fehler	Auth-Key abgelaufen → neuen Key generieren

□□ Best Practice

- Skript immer aus **interner Quelle** oder dem **internen Git-Repository** verwenden
- Keine Auth-Keys oder API-URLs in Klartext speichern
- Nach Installation prüfen, ob der Agent im Dashboard erscheint
- Bei Server-Deployments kann das Skript automatisiert

Knowledge Base

Aktive technische Knowledge-Base-Einträge aus Obsidian.

RustDesk in Tactical RMM integrieren

RustDesk in Tactical RMM integrieren

Diese Anleitung beschreibt die sichere Windows-Bereitstellung von RustDesk über Tactical RMM (TRMM). Ein eigener RustDesk-Server wird vorausgesetzt.

> **Status:** Dienstinstallation, ID-Ermittlung und URL-Aktion stammen aus einem Praxiseinsatz. Das alte Skript war jedoch auf RustDesk 1.2.6 und direkte Änderungen an `RustDesk2.toml` festgelegt. Der aktuelle Ansatz nutzt die dokumentierte Konfigurationszeichenfolge und muss mit jeder freigegebenen Client-Version pilotiert werden.

Empfohlener Ablauf

1. RustDesk als Dienst installieren.
2. Servereinstellungen mit einer RustDesk-Konfigurationszeichenfolge setzen.
3. RustDesk-ID in ein TRMM-Custom-Field schreiben.
4. Eine URL-Aktion **ohne Kennwort** hinterlegen.
5. Unbeaufsichtigten Zugriff über eine geeignete Kennwort- oder Kontorichtlinie absichern.

RustDesk empfiehlt für Server Pro den Custom Client Generator. Für OSS- oder RMM-Rollouts sind ein geprüftes PowerShell-Skript und `--config` die dokumentierte Alternative. Die TRMM-Beispielseite ist für Collector und URL Action nützlich, enthält aber eine veraltete, fest codierte Client-Version.

Voraussetzungen

- Die Konfigurationszeichenfolge stammt aus der RustDesk-Webkonsole oder aus **Settings → Network → Export Server Config**.
- Download-URL und SHA-256-Prüfsumme gehören zur selben freigegebenen Version.
- Das TRMM-Skript läuft als Administrator beziehungsweise `SYSTEM`.
- Geschützte Werte werden nicht im Skript protokolliert.

Installationsskript

Die Werte als TRMM-Parameter oder geschützte Variablen übergeben. Keine produktiven Servernamen, Kennwörter oder Tokens in die Knowledge Base eintragen.

```
param(
    [Parameter(Mandatory = $true)]
    [string]$DownloadUrl,

    [Parameter(Mandatory = $true)]
    [ValidatePattern('^[A-Fa-f0-9]{64}$')]
    [string]$ExpectedSha256,

    [Parameter(Mandatory = $true)]
    [string]$ConfigString
)

$ErrorActionPreference = 'Stop'
$workDir = Join-Path $env:ProgramData 'TacticalRMM\temp\rustdesk'
$installer = Join-Path $workDir 'rustdesk.exe'
New-Item -ItemType Directory -Force -Path $workDir | Out-Null

Invoke-WebRequest -Uri $DownloadUrl -OutFile $installer
$actualSha256 = (Get-FileHash -Path $installer -Algorithm SHA256).Hash
if ($actualSha256 -ne $ExpectedSha256) {
    throw 'RustDesk SHA-256 stimmt nicht überein.'
}

Start-Process -FilePath $installer -ArgumentList '--silent-install' -Wait

$service = Get-Service -Name 'RustDesk' -ErrorAction Stop
if ($service.Status -ne 'Running') {
    Start-Service -Name 'RustDesk'
}
```

```
$client = Join-Path $env:ProgramFiles 'RustDesk\RustDesk.exe'
& $client --config $ConfigString
if ($LASTEXITCODE -ne 0) {
    throw "RustDesk-Konfiguration fehlgeschlagen (Exit-Code $LASTEXITCODE)."
}

Restart-Service -Name 'RustDesk'
Remove-Item -Path $installer -Force
```

Nicht automatisch «latest» herunterladen: Version testen, freigeben, URL und SHA-256 pinnen und erst danach ausrollen.

RustDesk-ID erfassen

In TRMM ein Custom Agent Field `rustdeskid` vom Typ **Text** anlegen. Collector-Skript:

```
$ErrorActionPreference = 'Stop'
$client = Join-Path $env:ProgramFiles 'RustDesk\RustDesk.exe'
if (-not (Test-Path $client)) {
    throw 'RustDesk ist nicht installiert.'
}

$id = (& $client --get-id | Out-String).Trim()
if ([string]::IsNullOrWhiteSpace($id)) {
    throw 'RustDesk-ID konnte nicht ermittelt werden.'
}

$id
```

URL-Aktion

```
rustdesk://connection/new/{{agent.rustdeskid}}
```

Die Anwendung fordert danach die zulässige Authentisierung an.

> **Kein Kennwort in URL oder Custom Field:** Query-Strings, RMM-Felder, Skriptausgaben und Prozessargumente können in Logs oder Browser-Verläufen erscheinen. Kein dauerhaftes Kennwort als `?password=...` speichern. Für unbeaufsichtigten Zugriff vorzugsweise RustDesk-Pro-Konten/Adressbuch, zeitlich begrenzte Zugangsdaten oder einen geeigneten Secrets-Workflow einsetzen.

Abnahme

- `Get-Service RustDesk` meldet `Running`.
- ID-Server und öffentlicher Schlüssel sind korrekt.
- Die ID erscheint im TRMM-Custom-Field.
- Die URL-Aktion öffnet den ausgewählten Agenten.
- Direktverbindung und Relay-Fallback funktionieren aus einem externen Netz.
- Neuinstallation und Update wurden mit der gepinnten Version getestet.
- TRMM-Ausgabe, Skript und Custom Fields enthalten keine Kennwörter oder Tokens.

Quellen

- RustDesk: Client Deployment
- RustDesk: Client Configuration
- Tactical RMM: RustDesk-Integration

Quelle: Obsidian `Knowledge Base/TacticalRMM/RustDesk in Tactical RMM Integrieren.md` · Stand: 2026-10-12 · Automatisiert publiziert durch Hermes.

TRMM-Linux-Agent-Konzept

TRMM-Linux-Agent-Konzept

Kurzfassung

Tactical RMM (TRMM) dokumentiert inzwischen einen **Linux-Agenten im Beta-Status**. Code-signierte Agenten, einschliesslich Linux und macOS, stehen gemäss Hersteller ab Sponsor Tier 1 zur Verfügung. Dieser offizielle Weg ist für neue Installationen zu bevorzugen.

Ein organisationsintern gebauter Fork bleibt technisch möglich, verursacht aber dauerhafte Pflege-, Lieferketten- und Lizenzrisiken. Er ist keine gleichwertige Standardlösung.

> **Getestet:** Ein intern gebauter Agent auf Basis von Agent `v2.11.0` wurde auf Linux erfolgreich registriert; `tacticalagent.service` und die Mesh-Verknüpfung funktionierten. Dieser Befund gilt nur für den damaligen Fork und dessen Build-Pipeline.
>
> **Generisch/aktuell:** Die offiziellen Installationsschritte unten stammen aus der laufenden TRMM-Dokumentation. Funktionen und Dienstnamen sind nach jedem Agent-Update erneut zu prüfen, da der Linux-Agent als Beta bezeichnet wird.

Entscheid

1. **Standard:** Offiziellen Linux-Installer aus der eigenen TRMM-Instanz verwenden.
2. **Ausnahme:** Eigenbau nur, wenn der offizielle Bezug nachweislich nicht möglich ist und Security, Lizenz sowie Wartungsverantwortung geklärt sind.
3. **Nicht verwenden:** Ungepinnte Fremdskripte, Builds direkt aus `master` oder Live-Kompilation auf verwalteten Zielsystemen.

Offizielle Installation

1. In TRMM **Agents → Install Agent** öffnen.

2. Client, Site, Gerätetyp, Architektur, Ablaufzeit und **Linux** wählen.
3. Generiertes Shellskript herunterladen und sicher auf den Zielhost übertragen.
4. Ausführen:

```
chmod 700 ./rmm-<client>-<site>-<type>.sh
sudo ./rmm-<client>-<site>-<type>.sh
```

5. Nach erfolgreichem Rollout löschen, denn das Skript enthält ein zeitlich gültiges Enrollment-Token:

```
rm -f ./rmm-<client>-<site>-<type>.sh
```

Ein Linux-Deployment-Link wird laut TRMM-Dokumentation derzeit nicht angeboten. Das generierte Skript deshalb nur über einen geschützten, kurzlebigen Ablageort verteilen; niemals in ein öffentliches Repository oder eine öffentliche Knowledge Base hochladen.

Verifikation

```
sudo systemctl status tacticalagent.service --no-pager
sudo journalctl -u tacticalagent.service -n 100 --no-pager
```

Zusätzlich in TRMM prüfen:

- Host erscheint beim erwarteten Client und Standort.
- Letzter Check-in ist aktuell.
- Checks und Skripte laufen mit erwarteten Rechten.
- Mesh/Remote-Zugriff ist vorhanden, sofern mitinstalliert.
- Neustart stellt Agent und Mesh wieder her.

Funktionsumfang realistisch einordnen

Der Feldtest des internen Forks `v2.11.0` bestätigte Monitoring, Skriptausführung, Terminal und Mesh. In diesem Stand waren unter anderem Softwareinventar, grafisches Patch-Management, Service-Steuerung und Windows-spezifische Funktionen nicht oder nur eingeschränkt vorhanden.

Dies ist **keine aktuelle, vollständige Produktmatrix**. Vor dem Rollout muss die tatsächlich eingesetzte Agent-Version mit einem Testhost gegen die benötigten Funktionen geprüft werden.

Kontrollierter Eigenbau (Ausnahme)

1. Upstream-Tag und Commit pinnen; nie direkt aus `master` veröffentlichen.

2. Änderungen reviewen und Abhängigkeiten mit `go mod verify` prüfen.
3. In isolierter CI für die benötigten Architekturen bauen:

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -trimpath -ldflags="-s -w" -o rmmagent-linux-amd64
sha256sum rmmagent-linux-amd64 > SHA256SUMS
```

4. Artefakte und `SHA256SUMS` unveränderbar in einem zugriffsgeschützten Release-Repository ablegen.
5. Installer und Self-Update müssen die Prüfsumme **vor** dem Ersetzen der Binärdatei validieren.
6. Enrollment-Werte nur zur Laufzeit übergeben und nicht loggen.
7. Rollback, Update-Überwachung und regelmässige Upstream-Diffs betreiben.

> Eine SHA-256-Prüfsumme schützt nur dann vor Manipulation, wenn das Manifest über einen getrennt vertrauenswürdigen beziehungsweise signierten Kanal bezogen wird.

Sicherheits- und Lizenzgrenzen

- Die **Tactical RMM License v1.0 ist keine Open-Source-Lizenz**.
- Sie erlaubt die Nutzung zur Überwachung eigener und verwalteter Kundensysteme, enthält aber weitgehende Einschränkungen für kommerzielle Bereitstellung, SaaS, Managed Hosting, Installation/Konfiguration als Dienstleistung sowie abgeleitete Werke.
- Vor Verteilung eines Forks, Rebranding oder einem kommerziellen Angebot die konkrete Nutzung rechtlich und gegebenenfalls mit AmidaWare klären.
- Wird ein abgeleitetes Werk Dritten bereitgestellt, sind die Pflichten zur Bereitstellung des vollständigen Quellcodes und zum Copyright-Hinweis zu beachten.
- Diese Notiz ist keine Rechtsberatung.

Abnahme

- ☐ Offizieller Installer oder freigegebener, gepinnter Eigenbau verwendet
- ☐ Keine Tokens in Shell-History, Tickets, Logs oder Dokumentation
- ☐ Prüfsumme/Signatur validiert
- ☐ Check-in, Checks, Skripte und Remote-Zugriff getestet
- ☐ Neustart und Agent-Update getestet
- ☐ Funktionslücken dokumentiert

- [ ] Lizenz- und Wartungsverantwortung geklärt

Quellen

- Tactical RMM: Agent Installation – Linux Agent (beta)
- Tactical RMM: Code Signing
- Tactical RMM License v1.0
- Tactical RMM Agent (Upstream)

Quelle: Obsidian `Knowledge Base/TacticalRMM/TRMM-Linux-Agent-Konzept.md` · Stand: 2026-10-12 · Automatisiert publiziert durch Hermes.