USB Schnittstellen - Sicherheit

Teilsicherheitskonzept – CONBOTICS MalerRoboter

Inhalt

1.) Absicherung von USB-Schnittstellen gegen externe Zugriffe

Einleitung

Die Universal Serial Bus (USB)-Schnittstelle hat sich als universeller Standard für den Anschluss von Peripheriegeräten und den Datentransfer etabliert. Ihre Einfachheit und weite Verbreitung sind Segen und Fluch zugleich. Während sie den Arbeitsalltag erheblich erleichtert, stellt sie auch eines der signifikantesten und oft unterschätzten Einfallstore für Sicherheitsbedrohungen dar. Ein ungesicherter USB-Port ist eine offene Tür für Datendiebstahl, Malware-Infektionen und systemische Kompromittierungen.

Dieser Bericht bietet eine detaillierte Analyse der Bedrohungslandschaft und stellt ein umfassendes, mehrschichtiges Sicherheitskonzept vor, um Systeme und Netzwerke wirksam gegen Angriffe über USB-Schnittstellen zu schützen. Die empfohlenen Maßnahmen richten sich sowohl an Privatpersonen als auch an Unternehmen jeder Größe.

1.) Das Bedrohungspotenzial von USB-Schnittstellen

Um wirksame Schutzmaßnahmen zu ergreifen, ist ein Verständnis der spezifischen Angriffsvektoren unerlässlich.

- **Einschleusen von Schadsoftware (Malware):** Dies ist die bekannteste Bedrohung. Ein mit Malware (Viren, Trojaner, Ransomware, Keylogger) infizierter USB-Stick kann ein System kompromittieren, sobald er angeschlossen und dessen Inhalt ausgeführt wird. Oft geschieht dies durch die "AutoRun"-Funktion (in modernen Systemen meist entschärft) oder durch Social Engineering, das den Nutzer zum Öffnen einer infizierten Datei verleitet.

- **Datendiebstahl und Datenexfiltration:** Ein ungesicherter USB-Port ermöglicht es einem Angreifer mit physischem Zugang zum Gerät, innerhalb von Sekunden oder Minuten große Mengen sensibler Daten (z. B. Konstruktionspläne, Kundendaten, Finanzberichte) auf ein externes Speichermedium zu kopieren.
- **"BadUSB"-Angriffe:** Dies ist eine besonders raffinierte Angriffsform. Hier ist nicht eine Datei auf dem Stick bösartig, sondern die Firmware des USB-Geräts selbst wurde manipuliert. Das Gerät gibt sich gegenüber dem Betriebssystem nicht als Speicherstick, sondern als ein anderes Gerät aus, typischerweise als Tastatur (ein sogenanntes Human Interface Device, HID). Es kann dann unbemerkt Tastatureingaben simulieren, um Befehle auszuführen, Schadsoftware herunterzuladen oder Systemeinstellungen zu ändern.
- **"Juice Jacking":** Dieser Angriff zielt auf mobile Geräte ab, die an öffentlichen USB-Ladebuchsen (z. B. an Flughäfen oder in Cafés) aufgeladen werden. Manipulierte Ladeanschlüsse können neben dem Strom auch Daten übertragen, um das Gerät auszuspionieren oder Malware zu installieren.
- **Social Engineering durch "USB Baiting":** Hierbei werden absichtlich infizierte USB-Sticks an Orten wie Parkplätzen, in Kantinen oder Toiletten von Zielunternehmen "verloren". Die menschliche Neugier verleitet Mitarbeiter dazu, den Stick an ihrem Arbeitsrechner anzuschließen, um den Besitzer zu finden oder den Inhalt zu sichten, und lösen damit die Infektion aus.

2.) Hardwarebasierte Schutzmaßnahmen

Die erste Verteidigungslinie ist die physische Kontrolle über die Schnittstelle selbst.

- **2.1 Physische USB-Port-Blocker:** Diese kostengünstigen mechanischen Sperren werden in freie USB-Ports eingesetzt und können nur mit einem passenden Schlüsselwerkzeug entfernt werden. Sie bieten einen effektiven Basisschutz, indem sie das Anstecken jeglicher nicht autorisierter Geräte physisch verhindern. Ihr Einsatz ist besonders in öffentlich zugänglichen Bereichen oder an Terminals mit hohem Schutzbedarf zu empfehlen.
- **2.2 Hardwareverschlüsselte USB-Laufwerke:** Für den mobilen Datentransport ist die Verwendung von USB-Sticks und externen Festplatten mit integrierter Hardwareverschlüsselung (z. B. AES 256-Bit) zwingend erforderlich. Die Authentifizierung erfolgt oft über eine PIN-Eingabe auf einem Tastenfeld direkt am Gerät. Dies stellt sicher, dass die Verschlüsselung unabhängig vom Host-System ist und die Daten auch bei Verlust oder Diebstahl des Mediums geschützt bleiben. Für den Unternehmenseinsatz sollten FIPS 140-2-zertifizierte Geräte bevorzugt werden.

- **2.3 USB-Datenblocker ("USB-Kondome"):** Diese kleinen Adapter werden zwischen das USB-Kabel und die Ladebuchse gesteckt. Sie unterbrechen physisch die Datenleitungen (D+ und D-) im USB-Kabel und lassen nur die Stromleitungen durch. Dies ist eine simple, aber hocheffektive Methode, um sich an öffentlichen Ladestationen vor "Juice Jacking" zu schützen.

3.) Softwareseitige Kontrollmechanismen

Softwarelösungen ermöglichen eine flexible und granulare Steuerung der USB-Nutzung.

- ### 3.1 Konfiguration auf Betriebssystemebene:
    - **BIOS/UEFI-Einstellungen:** Die radikalste Methode ist die vollständige Deaktivierung der USB-Ports im BIOS/UEFI des Rechners. Dies bietet maximalen Schutz, aber auch keinerlei Flexibilität.
    - **Windows Gruppenrichtlinien (GPO):** In Windows-Domänen können Administratoren den Zugriff auf USB-Speichergeräte zentral steuern. Mögliche Konfigurationen umfassen:
        - **Vollständige Sperrung:** Jeglicher Zugriff auf USB-Speicher wird unterbunden.
        - **Schreibschutz:** Benutzer können Daten von USB-Geräten lesen, aber keine Daten darauf schreiben. Dies verhindert Datenabfluss.
        - **Whitelisting von Geräten:** Es kann eine Liste von genehmigten Geräten auf Basis ihrer Hardware-ID (VID/PID) erstellt werden. Nur diese explizit erlaubten Geräte (z. B. unternehmenseigene, verschlüsselte Sticks) können verwendet werden.
    - **macOS-Profile und Linux udev-Regeln:** Äquivalente Mechanismen existieren für andere Betriebssysteme, um ähnliche Kontrolllevel zu erreichen.
- ### 3.2 Dedizierte "Device Control"-Software (Endpoint Security):
  Professionelle Endpoint-Protection-Plattformen (EPP) oder Data-Loss-Prevention-Systeme (DLP) bieten noch weitreichendere Funktionen als GPOs. Vorteile sind:
    - **Zentrales Management:** Eine Konsole zur Verwaltung der Richtlinien für das gesamte Unternehmen.

    - **Detailliertes Logging und Reporting:** Jeder Versuch, ein Gerät anzuschließen (erfolgreich oder blockiert), wird protokolliert. Dies ist für Audits und die Analyse von Sicherheitsvorfällen unerlässlich.
    - **Kontextbasierte Richtlinien:** Regeln können basierend auf Benutzer, Abteilung, Netzwerkstandort oder Tageszeit variieren.
    - **Dateityp-Filterung:** Der Transfer bestimmter Dateitypen auf USB-Geräte kann blockiert werden.
- 3.3 Antiviren- und Anti-Malware-Lösungen: Jede professionelle Sicherheitssoftware sollte so konfiguriert sein, dass sie angeschlossene Wechselmedien automatisch und sofort ("On-Access-Scan") auf Bedrohungen überprüft. Dies ist eine reaktive, aber absolut notwendige Schutzebene.

Kapitel 4: Organisatorische und strategische Maßnahmen

Technik allein ist unzureichend. Die menschliche Komponente ist entscheidend für eine erfolgreiche Sicherheitsstrategie.

- 4.1 Erstellung und Kommunikation einer klaren IT-Sicherheitsrichtlinie**:** Jedes Unternehmen muss eine verbindliche Richtlinie zur Nutzung von Wechselmedien entwickeln. Diese muss klar definieren:
    - Welche Geräte sind erlaubt?
    - Ist die Nutzung privater Geräte gestattet? (Empfehlung: Nein)
    - Wie ist mit Medien von externen Partnern oder Kunden umzugehen? (z.B. Scan an einem isolierten Kiosk-System)
    - Wer sind die Ansprechpartner bei Fragen?
- 4.2 Kontinuierliche Sensibilisierung und Schulung der Nutzer**:** Die Mitarbeiter sind die erste Verteidigungslinie. Regelmäßige Schulungen müssen das Bewusstsein für die Gefahren schärfen und konkrete Verhaltensregeln vermitteln. Insbesondere die Bedrohung durch "USB Baiting" und Social Engineering muss immer wieder thematisiert werden.
- 4.3 Asset Management**:** Führen Sie ein Inventar aller ausgegebenen und genehmigten USB-Geräte. Dies ist die Grundlage für eine effektive Whitelisting-Strategie und hilft, den Überblick über potenziell unsichere Geräte zu behalten.

Fazit und Handlungsempfehlung

Die Sicherheit von USB-Schnittstellen erfordert einen mehrschichtigen Ansatz ("Defense in Depth"), der physische, technische und organisatorische Maßnahmen kombiniert. Eine einzelne Lösung bietet keinen ausreichenden Schutz.

Die empfohlene Strategie umfasst:

1. **Risikoanalyse:** Identifizieren Sie die Systeme mit dem höchsten Schutzbedarf.
2. **Basisschutz implementieren:** Konfigurieren Sie Antiviren-Scanner für das Scannen von Wechselmedien und nutzen Sie Betriebssystem-Richtlinien (GPO), um eine grundlegende Kontrolle (z. B. Schreibschutz) zu etablieren.
3. **Physische Sicherung:** Setzen Sie in sensiblen oder öffentlichen Bereichen USB-Port-Blocker ein.
4. **Sicheren Datentransport gewährleisten:** Schreiben Sie die ausschließliche Verwendung von hardwareverschlüsselten USB-Laufwerken für mobile Daten vor.
5. **Fortgeschrittene Kontrolle (optional):** In Hochsicherheitsumgebungen oder zur Verhinderung von Datenabfluss sollten dedizierte Device-Control-Lösungen implementiert werden.
6. **Mitarbeiter schulen:** Etablieren Sie ein fortlaufendes Schulungsprogramm als Fundament Ihrer Sicherheitskultur.

Durch die konsequente Umsetzung dieser Maßnahmen kann das Risiko eines Sicherheitsvorfalls über USB-Schnittstellen drastisch reduziert und die Integrität und Vertraulichkeit der Unternehmensdaten wirksam geschützt werden.

2.) USB Tests unter Linux

Das Testen von USB-Schnittstellen unter Linux lässt sich in zwei Hauptbereiche gliedern: die grundlegende Funktions- und Erkennungsprüfung sowie gezielte Sicherheitstests.

Teil 1: Grundlegende Funktions- und Erkennungstests

Bevor du Sicherheitstests durchführst, musst du wissen, wie das System auf normale Geräte reagiert. Diese Schritte sind sicher und rein diagnostisch.

Werkzeuge: Dein Terminal und ein *vertrauenswürdiger* USB-Stick.

Schritt 1: System-Logs in Echtzeit beobachten

Der erste Schritt ist immer, dem System beim Denken zuzusehen. Der Kernel protokolliert genau, was passiert, wenn ein USB-Gerät angeschlossen wird.

- **Befehl:** Öffne ein Terminal und gib ein:

Bash

```
dmesg -w
```

oder auf neueren Systemen:

Bash

```
journalctl -f
```

- **Aktion:** Lass den Befehl laufen und stecke dann deinen vertrauenswürdigen USB-Stick ein.
- **Beobachtung:** Du siehst nun live im Terminal, wie das Gerät erkannt wird. Achte auf Zeilen, die "USB" enthalten. Du solltest sehen:
    - Die Erkennung eines neuen USB-Geräts (`New USB device found...`).
    - Informationen zum Hersteller und Produkt (`idVendor, idProduct`).
    - Welcher Treiber geladen wird (z.B. `usb-storage`).
    - Welchem Gerätenamen es zugewiesen wird (z.B. `/dev/sdb`).
    - Fehlermeldungen, falls etwas nicht stimmt.
    -

Schritt 2: Angeschlossene USB-Geräte auflisten

- **Befehl:**

Bash

```
lsusb
```

- **Aktion:** Führe diesen Befehl aus, nachdem der Stick angeschlossen wurde.
- **Beobachtung:** Du erhältst eine Liste aller USB-Geräte. Dein Stick sollte hier mit einer Bus- und Gerätenummer sowie seiner Hersteller- und Produkt-ID (`xxxx:yyyy`) erscheinen. Dies bestätigt, dass das Gerät auf der USB-Ebene korrekt erkannt wird.

Schritt 3: Block-Geräte- und Mount-Status prüfen

- **Befehl:**

Bash

```
lsblk
```

- **Aktion:** Zeigt alle Block-Geräte an. Dein USB-Stick sollte als Gerät (z.B. `sdb`) mit einer oder mehreren Partitionen (z.B. `sdb1`) erscheinen.
- **Beobachtung:** Siehst du in der Spalte `MOUNTPOINT`, wo das Gerät eingehängt wurde (z.B. `/media/deinuser/STICKNAME)`? Wenn ja, funktioniert das automatische Einhängen (Automount).

Teil 2: Praktische Sicherheitstests

Hier wird geprüft, ob die Standardkonfiguration des Systems anfällig für gängige Angriffe ist.

Szenario A: Test auf Auto-Ausführung von Code

Ziel: Prüfen, ob das System das Ausführen von Programmen von einem Wechseldatenträger erlaubt.

1. **Vorbereitung:** Erstelle auf deinem USB-Stick ein einfaches Shell-Skript namens test.sh:

Bash

```
#!/bin/bash
```

echo "Sicherheitslücke: Code wurde ausgeführt!" > ~/Desktop/USB_TEST_ERFOLGREICH.txt

Mache die Datei ausführbar: chmod +x /pfad/zum/stick/test.sh.

2. **Test:**
    - Stecke den Stick an das Linux-System.
    - Prüfe die Mount-Optionen mit dem Befehl: mount | grep /dev/sd.
    - **Achte auf die Option noexec**. Wenn diese Option **fehlt**, ist das System potenziell verwundbar.
    - Versuche, das Skript im Terminal auszuführen: /media/deinuser/STICKNAME/test.sh.
3. **Ergebnis:** Wenn die Datei USB_TEST_ERFOLGREICH.txt auf deinem Desktop erscheint, hast du eine Sicherheitslücke gefunden. Eine sichere Konfiguration würde das Ausführen von Programmen auf Wechseldatenträgern standardmäßig blockieren (noexec).

Szenario B: Test auf Geräte-Emulation (BadUSB)

Ziel: Prüfen, ob das System einer als Tastatur getarnten Schadhardware blind vertraut.

1. **Vorbereitung:** Du benötigst ein BadUSB-Gerät (z.B. einen USB Rubber Ducky oder einen programmierten Arduino). Programmiere es mit einer harmlosen, aber sichtbaren Aktion für Linux:

    Payload-Beispiel: "Öffne ein Terminal und gib einen Text aus."

    - CTRL-ALT t // Terminal öffnen
    - DELAY 1000
    - STRING echo "System ist anfällig für BadUSB-Angriffe"
    - ENTER
2. **Test:** Stecke das BadUSB-Gerät an.
3. **Ergebnis:** Wenn sich das Terminal öffnet und der Befehl ausgeführt wird, fehlt dem System eine entscheidende Schutzmaßnahme: das Whitelisting von Geräten. Es vertraut jeder "Tastatur", die angeschlossen wird.

Szenario C: Physische Verwundbarkeit (Theoretischer Test)

Ziel: Bewusstsein für die Gefahr der physischen Zerstörung schaffen.

- **Methode:** Verwende **niemals** einen echten **USB-Killer** an einem Gerät, das du behalten möchtest. Der "Test" besteht darin, zu wissen, dass ungeschützte USB-Ports anfällig für Hochspannungsangriffe sind, die die Hardware permanent zerstören. Die einzige Gegenmaßnahme ist der physische Schutz.

Teil 3: Werkzeuge zur Härtung unter Linux

Wenn deine Tests Schwachstellen aufgedeckt haben, sind dies die wichtigsten Mechanismen zur Absicherung:

1. udev-Regeln**:** Das mächtigste Werkzeug zur Verwaltung von Geräten unter Linux. Du kannst in /etc/udev/rules.d/ eigene Regeln erstellen, um:
    - **Geräte zu blockieren oder zu erlauben (Whitelisting):** Erlaube nur Geräte mit einer bestimmten Hersteller- und Produkt-ID (ATTR{idVendor}=="...", ATTR{idProduct}=="...").
    - **Mount-Optionen zu erzwingen:** Sorge dafür, dass alle USB-Speicher automatisch mit noexec,nodev,nosuid gemountet werden.
2. Physische Port-Blocker**:** Die einfachste und oft effektivste Methode. Blockiere alle nicht benötigten USB-Ports mechanisch.

3.) UDEV Regeln der USB Anschlüsse

udev-Regeln sind ein extrem mächtiges Werkzeug unter Linux, um die Verwaltung von Geräten zu automatisieren. Wenn du mit USB-Geräten arbeitest, sind sie unerlässlich, um wiederkehrende Aufgaben zu automatisieren, die Sicherheit zu erhöhen oder einfach für mehr Ordnung zu sorgen.

Hier ist eine umfassende Erklärung, wie udev-Regeln für USB-Geräte funktionieren, von den Grundlagen bis zu praktischen Beispielen.

Was ist udev?

udev ist der Gerätemanager des Linux-Kernels. Seine Hauptaufgabe ist es, die Gerätedateien im Verzeichnis /dev zu verwalten. Wenn du ein USB-Gerät einsteckst, bemerkt der Kernel dies und übergibt die Information an udev. udev entscheidet dann basierend auf einem Satz von Regeln, was zu tun ist: wie die Gerätedatei benannt wird, welche Berechtigungen sie erhält oder ob ein bestimmtes Skript ausgeführt werden soll.

Eigene Regeln werden im Verzeichnis /etc/udev/rules.d/ abgelegt. Die Dateinamen sind wichtig: sie beginnen mit einer Zahl, die die Ausführungsreihenfolge bestimmt (z.B. 10-local.rules wird vor 99-my-usb.rules ausgeführt), und müssen auf .rules enden.

Aufbau einer udev-Regel

Eine udev-Regel besteht aus zwei Hauptteilen, die in einer einzigen Zeile stehen:

1. Abgleichs-Schlüssel (Matching Keys): Bedingungen, die erfüllt sein müssen, damit die Regel auf ein Gerät angewendet wird.
2. Zuweisungs-Schlüssel (Assignment Keys): Aktionen, die ausgeführt werden, wenn alle Bedingungen zutreffen.

Beispiel-Struktur: SUBSYSTEM=="usb", ATTRS{idVendor}=="1a2b", ATTRS{idProduct}=="3c4d", SYMLINK+="mein_geraet"

- SUBSYSTEM=="usb": Die Regel gilt nur für Geräte im USB-Subsystem.
- ATTRS{idVendor}=="1a2b" und ATTRS{idProduct}=="3c4d": Dies sind Abgleichs-Schlüssel. Die Regel greift nur, wenn ein USB-Gerät mit genau dieser Hersteller- (idVendor) und Produkt-ID (idProduct) angeschlossen wird.

- SYMLINK+="mein_geraet": Dies ist ein Zuweisungs-Schlüssel. Wenn die Regel zutrifft, wird zusätzlich zur normalen Gerätedatei (z.B. /dev/ttyUSB0) ein symbolischer Link namens /dev/mein_geraet erstellt.

Schritt 1: Informationen über dein USB-Gerät herausfinden

Um eine Regel schreiben zu können, musst du dein Gerät eindeutig identifizieren.

1. **Hersteller- und Produkt-ID finden:** Stecke dein USB-Gerät ein und gib im Terminal ein:

Bash

```
lsusb
```

Die Ausgabe sieht etwa so aus: Bus 001 Device 005: ID 0403:6001 Future Technology Devices International, Ltd FT232 Serial (UART) IC Hier ist die idVendor **0403** und die idProduct **6001**.

2. **Alle Attribute eines Geräts anzeigen:** Um noch mehr Details zu erhalten (z.B. die Seriennummer), benutze udevadm. Finde zuerst den Pfad deines Geräts (z.B. /dev/ttyUSB0):

Bash

```
udevadm info -a -n /dev/ttyUSB0
```

Dieser Befehl listet alle Attribute auf, die du in einer Regel verwenden kannst, z.B. ATTRS{serial} für die Seriennummer oder ATTRS{manufacturer}.

Schritt 2: Praktische Beispiele für udev-Regeln

Erstelle eine neue Regel-Datei mit sudo nano /etc/udev/rules.d/99-usb-custom.rules und füge die folgenden Beispiele ein.

Beispiel 1: Einen festen, persistenten Namen für ein Gerät vergeben

Problem: Ein USB-Seriell-Adapter ist mal /dev/ttyUSB0, mal /dev/ttyUSB1, je nachdem, in welcher Reihenfolge die Geräte eingesteckt werden. **Lösung:** Erstelle einen festen Link (Symlink).

Code-Snippet

```
# Regel für einen FTDI USB-Seriell-Adapter, erstellt einen Link /dev/mein_serial_adapter

SUBSYSTEM=="tty", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", ATTRS{serial}=="A1B2C3D4", SYMLINK+="mein_serial_adapter"
```

- **SUBSYSTEM=="tty"**: Wichtig, wir wollen die TTY-Schnittstelle, nicht das rohe USB-Gerät.
- **ATTRS{serial}=="A1B2C3D4"**: Die Verwendung der Seriennummer stellt sicher, dass die Regel nur für *genau dieses eine* Gerät gilt, selbst wenn du mehrere identische Modelle hast.
- **SYMLINK+="mein_serial_adapter"**: Erstellt den Link `/dev/mein_serial_adapter`. Deine Anwendung kann nun immer diesen festen Pfad verwenden.

Beispiel 2: Ein Skript beim Anschließen eines USB-Sticks ausführen

Problem: Immer wenn ein bestimmter Backup-USB-Stick angeschlossen wird, soll automatisch ein Backup-Skript starten. **Lösung:** Verwende den RUN-Schlüssel.

Code-Snippet

```
# Führt ein Backup-Skript aus, wenn ein bestimmter SanDisk USB-Stick angeschlossen wird

ACTION=="add", SUBSYSTEM=="block", SUBSYSTEMS=="usb", ATTRS{idVendor}=="0781", ATTRS{serial}=="1234567890", RUN+="/usr/local/bin/start_backup.sh"
```

- **ACTION=="add"**: Die Regel wird nur beim Anschließen ausgelöst.
- **SUBSYSTEM=="block"**: Wir reagieren, wenn das System ein Block-Gerät (eine Festplatte/Partition) erstellt.
- **RUN+="/usr/local/bin/start_backup.sh"**: Führt das angegebene Skript aus. **Wichtig:** Das Skript muss mit vollem Pfad angegeben werden und darf nicht zu lange laufen, da es udev blockieren kann. Für langlaufende Aufgaben sollte das Skript einen Hintergrundprozess starten.

Beispiel 3: Ein bestimmtes USB-Gerät blockieren (Security)

Problem: Ein nicht vertrauenswürdiges oder problematisches USB-Gerät soll vom System ignoriert werden. **Lösung:** Setze das `authorized`-Attribut auf `0`.

Code-Snippet

```
# Blockiert ein spezifisches USB-Gerät (z.B. eine unbekannte Webcam)
SUBSYSTEM=="usb", ATTRS{idVendor}=="abcd", ATTRS{idProduct}=="1234", ATTR{authorized}="0"
```

- **ATTR{authorized}="0"**: Weist den Kernel an, dieses Gerät nicht zu autorisieren. Es wird mit Strom versorgt, aber es werden keine Treiber geladen und keine Gerätedateien erstellt.

Schritt 3: Regeln anwenden

Nachdem du deine Regel-Datei gespeichert hast, musst du udev anweisen, die neuen Regeln zu laden.

Bash

```
# Lädt die Regel-Dateien neu
sudo udevadm control --reload-rules

# Wendet die neuen Regeln auf alle aktuell angeschlossenen Geräte an (optional)
sudo udevadm trigger
```

Am einfachsten testest du eine neue Regel, indem du das entsprechende USB-Gerät aus- und wieder einsteckst. udev wird dann die neue Regel automatisch anwenden.