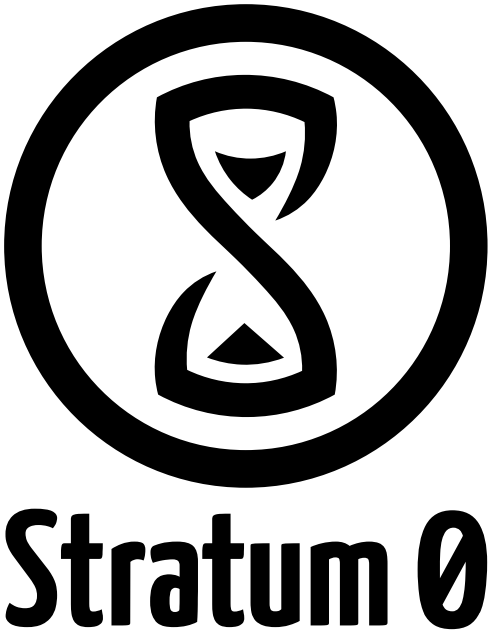
Linux-Dienste absichern mit systemd Sandboxing

Johannes Starosta

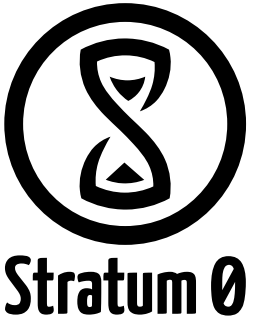
Mastodon: https://toot.kif.rocks/@joke

Matrix:
@joke:stratum0.org
@johannesst:datenburg.org



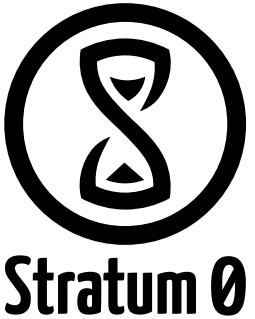
22.06.2025

Motivation



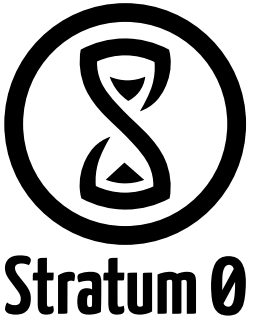
- Ich war mal auf einer Weiterbildung
- Dort gelernt:
  - Man kann mit systemd Diensten sehr gut Rechte wegnehmen
  - Relativ wenig Aufwand für low-hanging fruits
  - Funktioniert auch ohne AppArmor/SELinux und Co
- Wird erschreckend wenig gemacht
- Das will ich ändern

Kleine Umfrage



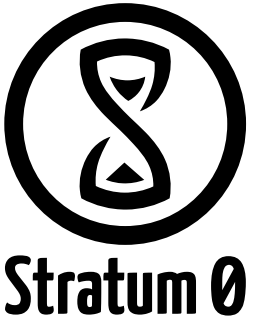
- Wieviele von euch
  - wissen, was systemd ist?
  - haben schon mal ein unit-file geschrieben oder bearbeitet?
  - sind Entwickler/in/Maintainer/in bei einen OpenSource Projekt?
  - sind systemd Entwickler/in oder Maintainer/in für systemd bei einer Distribution?

Agenda



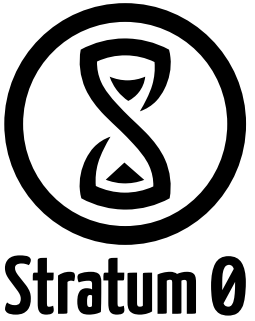
- Grundlagen (Namespaces, cgroups, Capabilities, Syscalls)
- Einschränkungen
- Beispieldienste
- Praktischer Teil mit euren Projekten :)
  - Ich gehe dafür davon aus, dass ihr ein aktuelles Linux verfügbar habt, ob auf euren Notebook, vserver oder als VM
- Slides und Beispiele:
- git clone https://github.com/johannesst/systemd-workshop-gpn23.git

Vorwarnung ;)



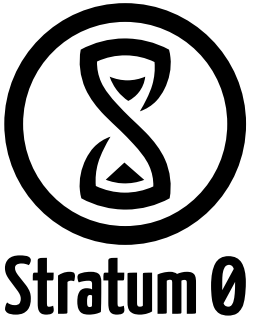
- Die folgenden Folien sind sehr textlastig
- Grund: Sollten innerhalb Workshops als Referenz dienen
- Verweisen auf Zeug zum Weiterlesen
- Also genauso, wie Vorträge NICHT sein sollten
- git clone https://github.com/johannesst/systemd-workshop-gpn23.git

Grundlagen



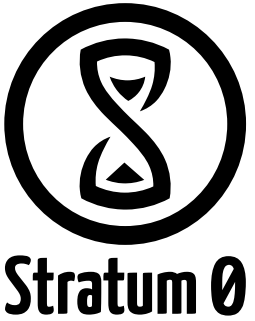
- Basiert auf gleichen Kram wie LXC, Docker und co (namespaces, control groups etc)
- Bei Ausführung in Containern muss darum Nesting aktiv sein (für die Incus und ProxmoxVE User ;))
- Hinreichend aktuelles systemd
  (entsprechender Code in systemd-analyze tauchte erstmals 2018 auf, wurde seitdem immer wieder erweitert)
- Debian Bullseye kann das, also auch Ubuntu 20.04/22.04
- In der Redhat Welt: Ab RockyLinux8 geht es auf jeden Fall
- Rest: Kein Plan, aber es sollte funktionieren ;)

Namespaces



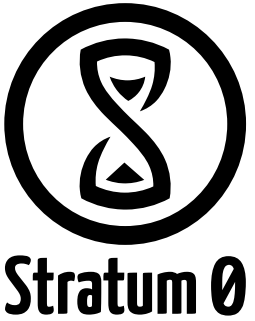
- Bilden Ressourcen (Prozesse, Geräte, Interfaces) des Hostsystems auf verschiedene Untermengen ab
- Lassen sich mit lsns(8) anzeigen
- nsenter(1) öffnet Shell in Namespace mit nsenter --all --target PID
- Mehr Informationen:
  - https://man7.org/linux/man-pages/man7/namespaces.7.html
  - https://man7.org/linux/man-pages/man1/nsenter.1.html
  - https://man7.org/linux/man-pages/man8/lsns.8.html
  - https://www.linux-magazin.de/ausgaben/2016/06/network-namespaces/
  - https://en.wikipedia.org/wiki/Linux_namespaces

cgroups



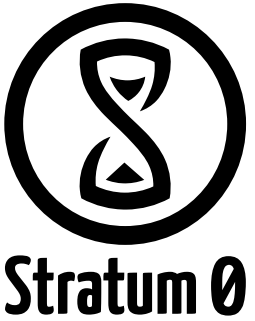
- Gibt es in Versionen v1 und v2, v1 sollte man nicht mehr nutzen (Support in podman, kubernetes und co deprecated oder nicht mehr vorhanden)
- Erlauben Zuweisung und Begrenzung von Ressourcen an Prozesse
- Beispiel: Prozess X darf maximal n MB Systemspeicher oder Netzwerkbandbreite belegen
- Ich habe davon nicht wirklich Ahnung, außer dass es von systemd und allen möglichen Containern (docker, podman, lxc) genutzt wird
- Wer mehr wissen will als ich:
  - https://wiki.archlinux.org/title/Cgroups
  - https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html
  - https://man7.org/linux/man-pages/man7/cgroups.7.html

Capabilities



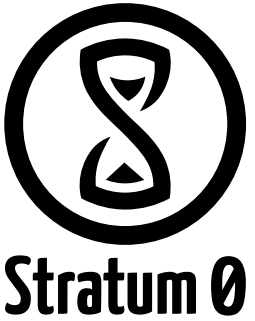
- Unter klassischen Unix-Systemen darf root alles
- Irgendwann fiel Leuten auf, dass das ein Problem ist
- Idee: Wir teilen die Vorrechte von root auf Teilmengen auf
- FreeBSD, MacOSX und NetBSD haben ähnliches (laut Stackexchange ;)
- Auf Dateiebene über setcap und getcap setz- und auslesbar
  - https://man7.org/linux/man-pages/man8/setcap.8.html
  - https://man7.org/linux/man-pages/man8/getcap.8.html
- Problem: Mittlerweile teilweise ähnlich überladen…
- Weiteres:
  - https://wiki.archlinux.org/title/Capabilities
  - https://man7.org/linux/man-pages/man7/capabilities.7.html

Syscalls und seccomp



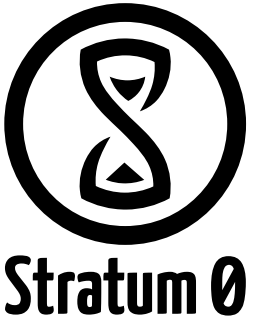
- Syscalls: API-Calls an Kernel, um z.B. eine Datei zu öffnen
- Seccomp: Mechanismus im Linux-Kernel um Prozessen Rechte zum Ausruf bestimmter Syscalls wegzunehmen
- Leider sehr komplex in der Benutzung auf API-Ebene (basiert auf bpf aka Berkely Packet Filter)
- Für BSD-User: OpenBSD hat dafür pledge, dessen API scheint handlicher zu sein
- Tagesaktuelle Liste mit allen Linux-Syscalls: https://gpages.juszkiewicz.com.pl/syscalls-table/syscalls.html
- Kerneldoku zu Seccomp: https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html

systemd.unit(5)



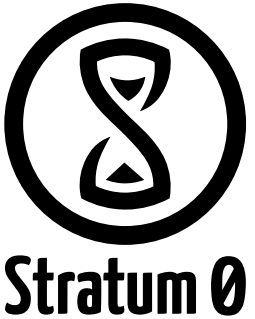
- Konfigurationsdatei, die durch systemd gemanagte Ressourcen definiert (Dienste, Sockets, Mounts, Devices, Timer...)
- Üblicherweise in /usr/lib/systemd/system (Distributionspakete) oder /etc/systemd/system/
- Inhalte können überschrieben/ergänzt werden durch sogenannte Overrides: Praktisch zum Testen, ohne „Original“ kaputt zu machen :)
- Für uns relevant:
  - systemd.socket(5) für Sockets (Inter-Prozess-Kommunikation)
  - systemd.service(5) für Dienste
  - systemd.timer(5) Um Dienst zu bestimmten Zeitpunkten/Intervallen auszuführen

Vorgehensweise



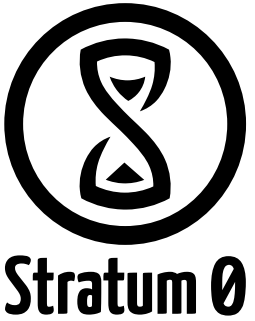
- systemd-analyze security dienstname.service gibt Übersicht, wie weit Dienste das nutzen
- Achtung: Das ist kein generisches Security-/Schwachstellenscanner etc!
- Es überprüft lediglich, wie weit die Sandboxing-Funktionen genutzt werden oder eben nicht
- Relevante manual pages:
  - systemd.analyze(1)
  - systemctl(1)
  - systemd.exec(5)
  - systemd.resource-control(5) (cgroup Schnittstelle!)

Legen wir los



- Neuen Dienst anlegen:
  systemctl edit --force --full dienstname.service
- Bestehenden Dienst bearbeiten:
  - systemctl edit --full dienstname.service
  - Lässt man –-full weg, bearbeitet man stattdessen overrides
- Dienst anzeigen und testen:
  - systemctl cat dienstname.service
  - systemd-analyze security dienstname.service
  - Systemctl enable dienstname.service
  - systemctl start dienstname.service

Woher weiß ich, was ich nehmen soll?

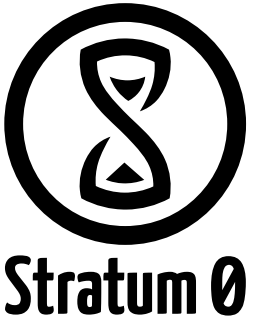


- Nette Menschen haben im Internet Vorschläge für sinnvolle Ausgangsbasis gemacht :)
- Wir gehen jetzt mal einen durch…
- git clone https://github.com/johannesst/systemd-workshop-gpn23.git
- `User,Group: Dienste sollten nicht als root laufen`
- `#Dienst und Kindprozesse dürfen keine neuen Privilegien anfordern`
  `NoNewPrivileges=yes`
- `# Dienst kriegt eigenes /tmp und /var/tmp zur Laufzeit, wird nach`

  `# stoppen wieder weggeräumt. Nicht sichtbar für andere Prozesse`

  `PrivateTmp=yes`
- `# Dienst kriegt eigenes /dev ohne Zugriff auf Hardware`
  `(/dev/hda /dev/mem /dev/tty etc), aber mit Pseudo-TTY, /dev/zero,`
  `/dev/random etc`
  `PrivateDevices=yes`

Woher weiß ich, was ich nehmen soll?

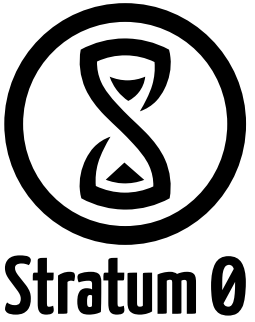


- Nette Menschen haben im Internet Vorschläge für sinnvolle Ausgangsbasis gemacht :)
- Wir gehen jetzt mal einen durch…
- `User,Group: Dienste sollten nicht als root laufen`
- `#Dienst und Kindprozesse dürfen keine neuen Privilegien anfordern`
  `NoNewPrivileges=yes`
- `# Dienst kriegt eigenes /tmp und /var/tmp zur Laufzeit, wird nach`

  `# stoppen wieder weggeräumt. Nicht sichtbar für andere Prozesse`

  `PrivateTmp=yes`
- `# Eigener Netzwerk-Namespace für Dienst,loopback only, Dienst`
  `# kann dann nicht auf Netzwerk vom Host zugreifen`
  `PrivateNetwork=yes`
- `# Netzwerkdienste brauchen im Regelfall nur IPv4, IPv6,`
  `# Unix-Sockets und envtl. die Kernel-/Userspace-Kommunikation`
  `RestrictAddressFamilies=AF_UNIX AF_INET AF_INET6 AF_NETLINK`

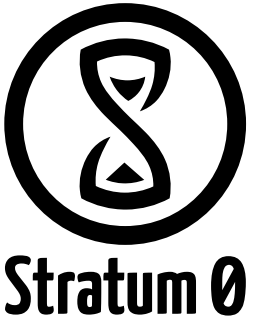
Woher weiß ich, was ich nehmen soll?




- ```
  # Dienst kriegt eigenes /dev ohne Zugriff auf Hardware
  # (/dev/hda /dev/mem /dev/tty etc), aber mit
  # Pseudo- TTY, /dev/zero, /dev/random etc
  PrivateDevices=yes
  ```

- ```
  # Nur Zugriff auf Standardpseudodateien erlaubt, u.A:
  # /dev/null, /dev/zero, /dev/full, /dev/random, und
  # /dev/urandom.
  # Alternative Optionen:
  # DevicePolicy=strict Nur explizit mit
  # DeviceAllow zugelassene Devices/Device-Gruppen
  # werden erlaubt
  # DevicePolicy=Auto Standardpolicy, sofern nichts
  # anderes oder DeviceAllow angegeben: Alle erlaubt
  DevicePolicy=closed
  ```

Woher weiß ich, was ich nehmen soll?



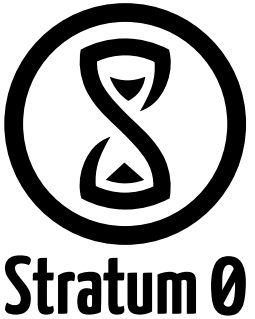

- ```
  # Sämtliche Verzeichnisse, außer /dev/, /proc/ und /sys/ stehen nur lesend zur Verfügung
  # Alternative Werte:
  # true /usr/ /boot und /efi readonly mounten
  # full Wie true plus /etc readonly
  ProtectSystem=strict
  # /home, /root und /run/user stehen nur lesend
  # zur Verfügung
  ProtectHome=read-only
  # Alternative Werte:
  # true  /home,/root und /run/user werden inaccessible für Dienst
  # tmpfs /home, /root und /run/user werden auf
  # tmpfs (quasi RAM-Disk) gemappt
  ```

- ```
  # Dienst kriegt eigenes /tmp und /var/tmp zur Laufzeit
  # wird nach  stoppen wieder weggeräumt. Nicht sichtbar für andere Prozesse
  PrivateTmp=yes
  ```

- ```
  # Falls man dann doch auf Verzeichnisse  zugreifen will
  # oder weiter Einschränkungen setzen:
  ReadWritePaths=, ReadOnlyPaths=, InaccessiblePaths=, ExecPaths=, NoExecPaths=
  ```

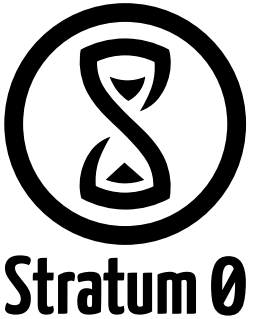
- ```
  # Falls man vorher inaccessible gemachte Verzeichnisse wieder verfügbar
  # machen will
  BindPaths=, BindReadOnlyPaths=
  ```

Woher weiß ich, was ich nehmen soll?



- `# Ist der Dienst ein container manager?`
  `# Nein, dann braucht er auch keinen Schreibzugriff auf`
  `# control groups`
  `ProtectControlGroups=yes`
- `# Dienste müssen normalerweise keine Kernelmodule laden`
  `ProtectKernelModules=yes`
- `#Die meisten Dienste brauchen keinen Schreibzugriff auf`
  `#/proc/sys/, /sys/, /proc/sysrq-trigger,`
  `# /proc/latency_stats, /proc/acpi,`
  `# /proc/timer_stats, /proc/fs und /proc/irq`
  `# Also: Nur Lesezugriff erlauben!`
  `ProtectKernelTunables=yes`
- `# Muss der Dienst an den Namespaces rumspielen? Meistens nicht`
  `RestrictNamespaces=yes`
- `# Die meisten Dienste brauchen kein RealtimeScheduling`
  `RestrictRealtime=yes`
- `# Der Dienst darf nie das set-user-ID (SUID) oder set-group-ID`
  `(SGID) setzen`
  `RestrictSUIDSGID=yes`

Woher weiß ich, was ich nehmen soll?



- Folgendes klappt auch meistens:

```
# Verbietet Speicherbereiche als gleichzeitig
# beschreib- und ausführbar zu markieren
# Achtung: OpenJDK (und ähnlich arbeitende Runtimes)
# kommen damit nicht zurecht
MemoryDenyWriteExecute=yes
```

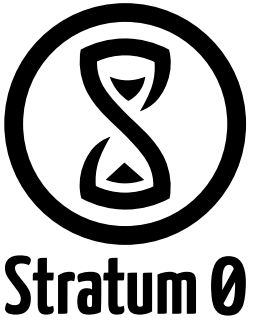
```
# Wird von systemd.exec(5) für die meisten Dienste
# empfohlen. Dienst sieht dann keine Prozesse anderer
# User in /proc
ProtectProc=invisible
```

```
# Mappe alle User und Gruppen auf nobody außer
# root und den Werten
# hinter User und Group (z.B. www-data)
PrivateUsers=yes
```

```
# Dienst soll nicht an Hardwareuhr, den Hostnamen oder den
# Kernellogbuffer rumfummeln
```

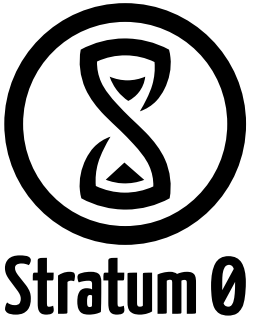
```
ProtectClock=yes
ProtectHostname=yes
ProtectKernelLogs=yes
```

Capabilities einschränken



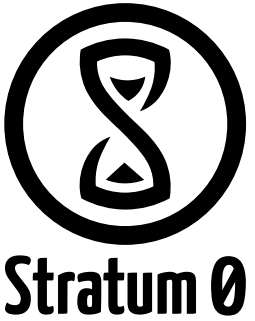
- capabilities(7) sind seit 2.2 im Linux Kernel
- Hintergrund: Klassische Trennung zwischen priviligierten (root) und unpriviligierten (alle anderen) Prozessen nicht so dolle
- Capabilities erlauben weitere Granualität
- Im system vorhandene anzeigen:
  - systemd-analyze capabilities
- Systemd erlaubt es den Diensten explizit welche wegzunehmen und zu genehmigen:
  - AmbientCapabilities Zusätzliche erlauben
  - CapabilityBoundingSet: Capabilites wegnehmen

SystemCalls einschränken



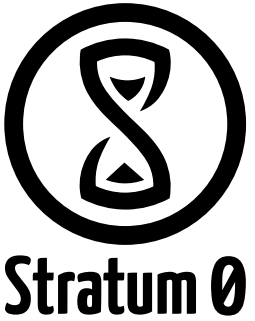
- Man kann den Diensten explizit das Ausführen bestimmter Systemcalls erlauben oder verbieten
- Problem: Woher weiß ich, welche ich nehmen soll?
- Gibt glücklicherweise ein paar sets mit (mehr oder weniger) sinnvollen Defaults:
  - @clock für Änderungen an der Systemzeit
  - @file-system analog für alles mit Dateizugriffen
  - @obsolete Obsolete, seltene oder gar nicht implementierte Syscalls
  - @resources Syscalls für Änderungen am Scheduling und Ressourcenlimits
  - @privileged Alle die Superuser-Rechte benötigen
  - @system-service Wichtigster Kram für Dienste

SystemCalls einschränken



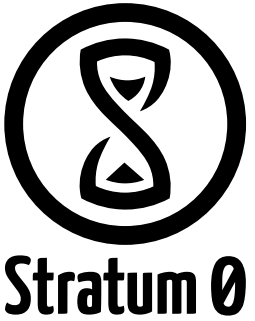
- Im Idealfall fängt man damit an, dass man @system-service erlaubt und @privileged sowie @resources verbietet
- Wenn man Glück hat, geht noch alles (dann kann weitere Sachen abschalten), wenn nicht muss man halt nach und nach wieder Sachen erlauben ;)
- SystemCallArchitectures=native schränkt es auf native CPU-Architektur ein, alternativ kann man angeben, welche man haben will
- Als Entwickler/in sollte man davon eigentlich (Tm) eine genaue Vorstellung haben
- Liste ist nicht vollständig, genaueres:
  - sudo systemd-analyze syscall-filter Zeigt alle vordefinierten Sets samt Syscall
  - systemd.exec(5) https://man7.org/linux/man-pages/man5/systemd.exec.5.html

systemd-Socket-Activation



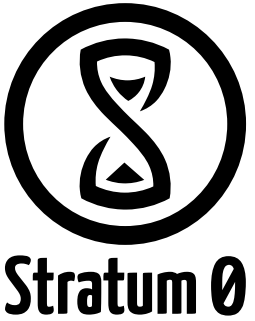
- dienst.socket definiert Sockets/Ports, die der Dienst nutzen soll soll
- dienst.service dann den eigentlichen Dienst
- Für alte Menschen (wie mich): Analog zu inetd und co ;)
- Vorteil: Dienst kann PrivateNetworks=true haben
- Beispiele:
  - Carsten Strotmann: Gophermoon https://strotmann.de/
  - Paul Brown: Simple echo server https://www.linux.com/training-tutorials/end-road-systemds-socket-units/

Weitere Nettigkeiten (die ich noch nicht ausprobiert habe...)



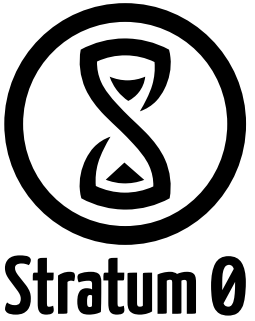
- DynamicUser: Erzeugt zur Laufzeit User/Gruppe, die nur im RAM und nur für den Dienst existieren
- PrivateNetwork=true Dienst hat eigenen Network namespace mit nur einem loopback Interface
- RootDirectory: chroot
- RootImage: mountet block oder loopback device und chrootet dann darin
- MountImages: Analog zu RootImage nur ohne loopback
- BindPaths=, BindReadOnlyPaths= bind mounts
- SELinuxContext=,AppArmorProfil= AppArmor-Profil oder SELinuxContext für Dienst setzen
- systemd-container mit systemd.nspawn
- Diverse andere Sachen ;)

Einschränkungen



- Je nachdem, welche Optionen sonst aktiv sind, wird der angestrebte Effekt nicht erreicht (siehe systemd.exec(5) für Details)
- Fummeln an gleichen Sachen herum wie System (lxc) und Application (OCI)-Container, muss in ihnen darum nicht funktionieren (nested in Containereinstellung sollte helfen)
- Kann nur solange funktionieren, wie keine Sicherheitslücken in systemd und/oder im „Hintergrund“ genutzen Kernelkomponenten drin sind

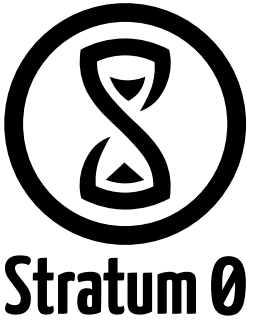
Weitere Beispiele „zur Inspiration“



- Alle Dienste von systemd selbst, z.B. systemd-journald oder systemd-timesyncd
- Smallstep (ein ACME-Server, falls eine eigene CA betreiben wollt/müsst): https://github.com/smallstep/certificates/blob/master/systemd/step-ca.service

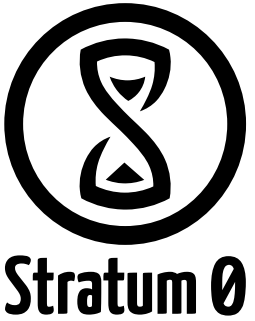
- Kann man sich mit systemd-analyze security dienstname.service und systemctl cat dienstname.service anzeigen lassen
- Die auf den Slides zu Sockets erwähnten :)

Zum Weiterlesen



- Manual pages:
  - systemd.analyze(1)
  - systemctl(1)
  - systemd.exec(5)
  - systemd.resource-control(5)
  - capabilites(7)
- Internet (Blogposts und co):
  - Systemd Hardening allgemein:
    - https://www.flashsystems.de/articles/systemd-hardening/
    - https://www.linuxjournal.com/content/systemd-service-strengthening
  - Erklärung der Optionen für Capabilities:
    - https://unix.stackexchange.com/q/580597
- Vorschlag für Standard-Pattern plus Erklärungen: https://gist.github.com/ageis/f5595e59b1cddb1513d1b425a323db04
  - Gibt noch mehr Erklärungen plus Beispiele, sowohl bei github als auch sonstwo
    →Google systemd hardening
- Beispiel für container mit systemd-nspawnd: Ein Gopher-Server in LUA https://strotmann.de/archive.html → Dort die Postings zu gophermoon

Praktischer Teil ;)



- Startet eure Linux-VM/Umgebung
- Habt ihr gerade ein Projekt? Dann versucht dafür ein neues service-File zu schreiben
- Falls nein: Nimmt den Python-Webserver aus den Beispielen
- https://github.com/johannesst/systemd-workshop-gpn23.git
- Simplehttp.service, simplehttp-pattern.services und simplehttp-final.service sind als Vorlagen gedacht (sind wir vorhin durchgegangen)
- Neuen Dienst anlegen:
  systemctl edit --force --full dienstname.service
- Bestehenden Dienst bearbeiten:
  - systemctl edit --full dienstname.service
  - Lässt man –-full weg, bearbeitet man stattdessen overrides
- Dienst anzeigen und testen:
  - systemctl cat dienstname.service
  - systemd-analyze security dienstname.service
  - systemctl start dienstname.service

Slides und Beispiele:
https://github.com/johannesst/systemd-workshop-gpn23.git

Danke für eure Aufmerksamkeit!

Johannes Starosta
Mastodon: https://toot.kif.rocks/@joke
Matrix:
@joke:stratum0.org
@johannesst:datenburg.org

22.06.2025 GPN 23 Karlsruhe