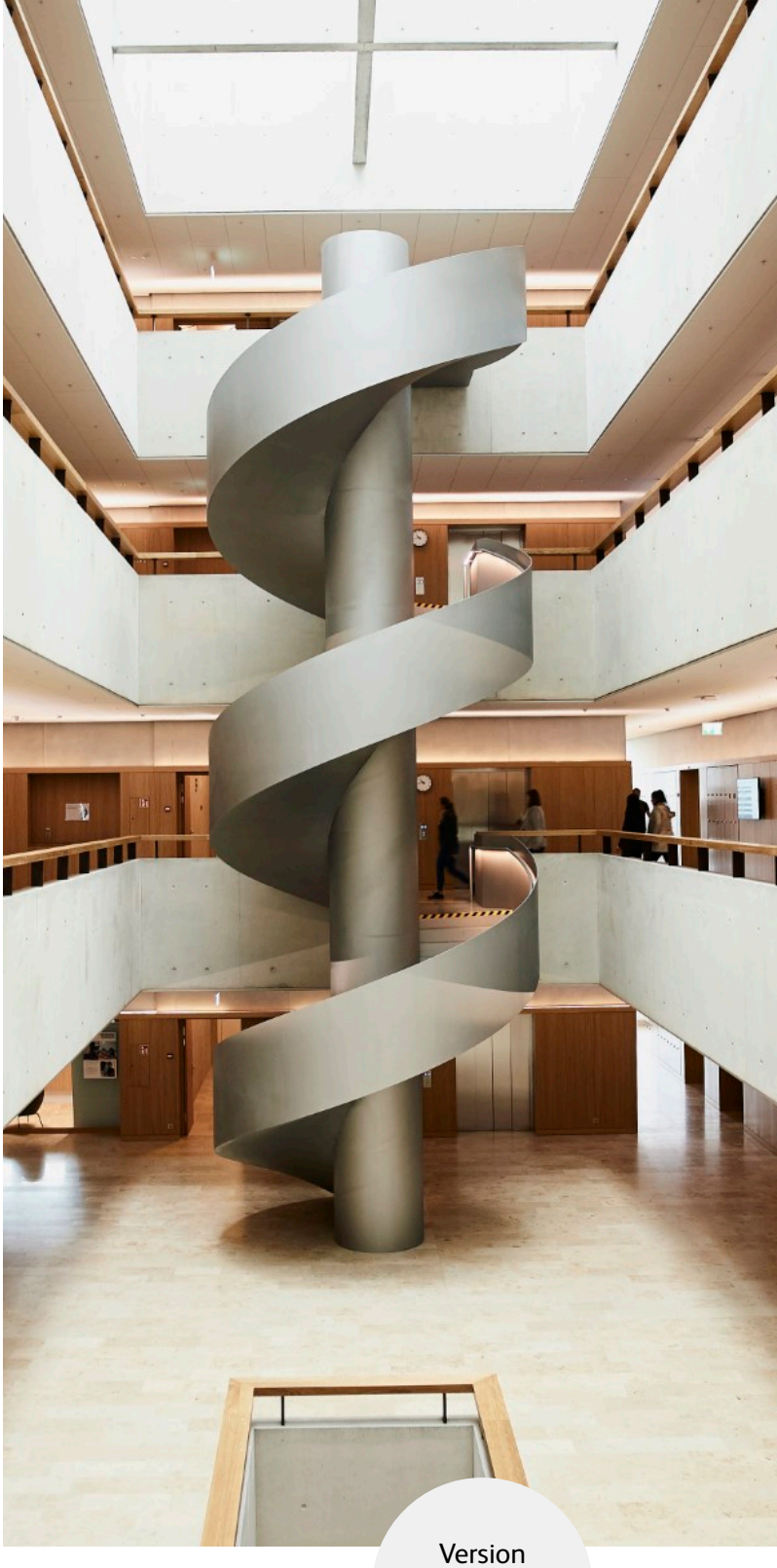
Hochschule Luzern
Modul I.BA_OSSEC



Laborübung

Isolation

Version
0.0.0-content

HSLU Hochschule Luzern

I. Allgemeine Informationen

Name:

Gruppe:

Bemerkungen:

Autoren

S. Miescher, J. Zavrlan, M. Useini

Version

Letzte Änderung: 2024-05-06
Dokument Version: 0.0.0-content (#38447f7)
Template Version: 1.2.0 (#6e0dbad)

II. Inhaltsverzeichnis

III. Vorwort

Feedback

Mit Ihrer Mithilfe kann die Qualität des Versuches laufend den Bedürfnissen angepasst und verbessert werden.

Falls in diesem Versuchsablauf etwas nicht so funktioniert wie es beschrieben ist, melden Sie dies bitte direkt dem Laborpersonal oder erwähnen Sie es in Ihrem Laborbericht oder Protokoll. Behandeln Sie die zur Verfügung gestellten Geräte mit der entsprechenden Umsicht.

Bei Problemen wenden Sie sich bitte ebenfalls an das Laborpersonal.

Legende

In den Versuchen gibt es Passagen, die mit den folgenden Boxen markiert sind. Diese sind wie folgt zu verstehen:

Wichtig

Dringend beachten. Was hier steht, unbedingt merken oder ausführen.

Aufgabe III.1

Beantworten und dokumentieren Sie die Antworten im Laborprotokoll.

Hinweis

Ergänzender Hinweis / Notiz / Hilfestellung.

Information

Weiterführende Informationen. Dies sind Informationen, die nicht zur Ausführung der Versuche benötigt werden, aber bekannt sein sollten.

Story

Hierbei wird die Geschichte vermittelt, die in den Versuch einleitet oder den Zweck des Versuches vorstellt.

Zielsetzung

Lernziele, die nach dem Bearbeiten des Kapitels erfüllt sein sollten.

Erkenntnis

Wichtige Erkenntnisse, die aus dem Versuch mitgenommen werden sollten.

1. Übungsaufgaben

1.1. Isolations-Technologien

Dieses Dokument beinhaltet die Versuchsanleitung für die Durchführung des Laborversuches zu Isolations-Technologien.

Bei Fragen zur Versuchsanleitung wenden Sie sich bitte direkt an das Laborpersonal.

2. Abkürzungsverzeichnis

Abkürzung	Beschreibung
RAM	Random Access Memory (Hauptspeicher)
VM	Virtuelle Maschine
COW	Copy-on-Write

3. Vorbereitung

3.1. Tasks

Zur Vorbereitung für die Laborübung sind folgende Tasks im vornherein zu erledigen.

- Lesen Sie die Manpage von ls und beantworten Sie die folgende Fragen.

```
man ls
```

3.2. Theoriefragen

Mit welchem Parameter zeigt ls Informationen im «long mode» an? Wo in der Ausgabe werden Benutzer und Gruppe einer Datei angezeigt?

3.3. Benötigte Mittel

Für diesen Versuch wird ein SSH-Zugang zur Laborübungs-VM benötigt.

Sämtliche Aktionen lassen sich über die Kommandozeile durchführen, die VM verfügt nicht über eine grafische Oberfläche.

3.4. Abgabe

Geben Sie das PDF mit ausgefüllten Formularfeldern innerhalb einer Woche über ILIAS ab.

4. chroot

Eine der einfachsten und gängigsten Möglichkeiten zur Isolation ist chroot. Der unix-Befehl bedeutet «change root directory», also einen Wechsel des Wurzelverzeichnisses. Damit lässt sich eine Umgebung schaffen, in der Programme in ein Verzeichnis eingesperrt werden können. Aus Sicht des Programms befindet sich das Wurzelverzeichnis / des Dateisystems im chroot-Verzeichnis. Das chroot-Verzeichnis wird oft auch «jail» genannt.

Wie bringt uns das mehr Sicherheit? In Unix-Systemen ist / das höchste Verzeichnis. Höher als das kann man nicht gehen, ein Wechsel ins Über-Verzeichnis von / geht wieder zu /. Ein Programm, das in einer chroot-Umgebung eingesperrt ist, kann also nicht auf Dateien zugreifen, die ausserhalb des Jails liegen.

Somit ist chroot eine einfache Variante zur **Isolation des Dateisystems**: Das isolierte Programm kann nicht auf Dateien ausserhalb der Isolation zugreifen. Normale Programme, die ausserhalb der Isolation laufen, haben aber sehr wohl Zugriff auf die Dateien im Jail – diese Programme sind ja auch nicht isoliert.

Der chroot-Befehl hat folgende Syntax:

```
chroot *NEWROOT* *COMMAND*
```

Als command kann ein normales Programm oder auch eine Shell genommen werden. Probieren Sie dies aus:

- Als erstes erstellen Sie ein neues leeres Verzeichnis:

```
mkdir jail
```

Sie können chroot **nur mit root-Rechten ausführen**, also Befehle immer mit sudo ausführen!

4.1. Triviale bash-shell

- Nun möchten Sie die bash-Shell in diesem Verzeichnis eingesperrt ausführen:

```
sudo chroot jail /bin/bash
```

Das funktioniert nicht, stattdessen wird ein Fehler angezeigt: chroot: failed to run command '/bin/bash': No such file or directory.

Was ist das Problem? Innerhalb des chroot-Verzeichnisses gibt es kein /bin/bash! Wenn das Wurzelverzeichnis geändert wird, muss auch das auszuführende Programm darin vorkommen.

4.2. Kopierte bash-shell

- Sie können nun innerhalb des Verzeichnisses eine Kopie von /bin/bash anlegen:

```
mkdir jail/bin
```

```
cp /bin/bash jail/bin/
```

```
sudo chroot jail /bin/bash
```

Das funktioniert immer noch nicht, und der Fehler bleibt gleich: chroot: failed to run command '/bin/bash': No such file or directory.

Was Ihnen die eher dürftige Fehlermeldung nicht sagt: Innerhalb von jail existiert jetzt sehr wohl /bin/bash, was Sie mit ls auch überprüfen können. Das Problem ist jetzt, dass bash eine Handvoll *shared libraries* benötigt, um zu funktionieren. Unter Windows wären das fehlende dll-Dateien, unter GNU/Linux fehlen so-Dateien.

- Diese können Sie mit ldd anzeigen lassen:

```
ldd jail/bin/bash
```

Sie sehen eine Liste von vier bis fünf fehlenden Libraries. Jetzt könnten Sie diese ebenfalls manuell ins jail-Verzeichnis kopieren, und die Abhängigkeiten dieser Libraries auch, und so weiter. Einfacher ist es aber, eine Shell zu nehmen, die keine weiteren Libraries braucht.

4.3. Busybox

Dafür bietet sich Busybox an, eine minimale Implementation der Unix-Standard-Befehle inklusive Shell. Busybox bietet die Shell und die Standard-Befehle in einem einzigen Programm. Das erspart auch gleich das Kopieren dieser Befehle in das jail.

Dieses Programm liegt uns in einer statischen Version vor, kommt also komplett ohne externe Libraries aus.

- Überprüfen Sie dies gleich selbst:

```
ldd /bin/busybox
```

Was ist die Ausgabe dieses Befehls?

Hier meldet uns ldd offenbar, dass es sich nicht um ein dynamisch gebaute ausführbare Datei handelt. Dynamisch sind Programme, die externe Shared Libraries benutzen, wie eben bash.

Diesmal sollte es also funktionieren. Busybox enthält wie erwähnt diverse Befehle und möchte deshalb noch wissen, welchen davon wir ausführen wollen – nämlich die Shell, also sh:

```
cp /bin/busybox jail/bin/
```

```
sudo chroot jail /bin/busybox sh
```

Und tatsächlich bekommen wir eine root-Shell, die ins jail-Verzeichnis eingesperrt ist.

- Schauen Sie nun mit ls alle vorhandenen Dateien und Verzeichnisse auf dem System an, beginnend im Wurzelverzeichnis /.

Welche Dateien und Verzeichnisse existieren innerhalb der chroot-Umgebung? Wie viele sind das total?

4.4. Unprivileged User

Wie Sie gemerkt haben, wird chroot und somit auch die Busybox-Shell mit sudo gestartet, die Shell hat also root-Rechte.

- Sie können dies mit dem id-Befehl innerhalb von Busybox verifizieren:

```
id
```

Die User-ID (uid) 0 bedeutet root.

Wir verwenden hier id statt whoami, weil innerhalb von chroot die Übersetzung von User-ID zu Username nicht gemacht werden kann, da die Datei /etc/passwd fehlt, in der Usernamen stehen.

Verlassen Sie nun die Chroot-Shell wieder mittels exit oder Ctrl-D.

Da wir chroot als Sicherheitsfeature verwenden, sollte auch die chroot-Umgebung nicht als root-user laufen, sonst haben wir potenziell neue Sicherheitsprobleme. Chroot löst dies trivial mit dem userspec-Parameter.

- Starten Sie die Chroot-Shell im Kontext des labadmin-Benutzers:

```
sudo chroot --userspec=labadmin:labadmin jail /bin/busybox sh
```

Nun sollten Sie in der Shell die uid des labadmin-Users sehen.

Was ist die angezeigte uid?

Chroot braucht so zwar noch root-Rechte zum Starten, wechselt aber nach dem Aufsetzen seine eigenen Berechtigungen auf den labadmin-User und **läuft von da an als labadmin**. Dieser Wechsel kann nicht rückgängig gemacht werden und ist somit sicher. Auch das eingesperrte Programm (busybox) wird dann als labadmin gestartet. Diesen Vorgang nennt man **privilege drop**.

Zusätzliche Erkenntnis: Die User-IDs innerhalb und ausserhalb des chroot-Jails sind identisch. Auch wenn er nicht mehr auf externe Dateien zugreifen kann, hat der labadmin-User im Jail doch die gleichen Berechtigungen wie der labadmin-User ausserhalb. Somit sind **Benutzer nicht isoliert**.

- Verlassen Sie die chroot-Umgebung wieder (Ctrl-D oder exit)

5. Container

Der Nutzen von chroot beschränkt sich auf eine Isolation des Dateisystems. Damit ist schon viel gewonnen, denn unter GNU/Linux-ähnlichen Betriebssystemen hält sich das Prinzip «alles ist eine Datei».

Container bieten zusätzlich eine **Isolation von Benutzer-Accounts, Prozess-Namespace und Netzwerk**. Wenn Container-Lösungen die Funktionalität von cgroups (ein Feature des Linux-Kernels) verwenden, lassen sich ausserdem Limits auf CPU, Hauptspeicher, Festplattenzugriff und Netzwerkdurchsatz konfigurieren.

5.1. LXC

Linux Containers (LXC) ist eine Implementation von Containerisierung unter Linux. LXC bietet eine Schnittstelle zu den verschiedenen vorhandenen Kernel-Interfaces für Isolation und Virtualisierung, um diese Funktionalität bereitzustellen; im Gegensatz zu anderen Container-Lösungen wie OpenVZ sind dafür keine Kernel-Patches oder separate Kernel-Module nötig. Diverse andere Container-Software wie LXD und ältere Versionen von Docker bauen auf LXC auf.

Als Beispiel können Sie mit LXC eine isolierte Verzeichnisstruktur erstellen und darin eine frische Betriebssystem-Installation vornehmen. Dabei sind diverse Dinge anders als bei einer normalen Installation – beispielsweise teilt sich das Container-Betriebssystem den laufenden Kernel mit dem Host, und idealerweise sollte nur das nötigste Minimum installiert werden. LXC kommt mit einer Reihe von Scripts (LXC nennt diese «Templates»), die diesen Prozess automatisieren.

- Mit dem mitgelieferten Template «debian» können Sie nun einen Test-Container erstellen:

```
sudo lxc-create --name test --template debian -- --release bookworm
```

Den Namen können Sie auch anders wählen, bei den weiteren Befehlen sollten Sie dann wiederum den gleichen Namen verwenden. Mit dem separaten -- werden Parameter zum Template abgetrennt. Hier übergeben Sie als Argument den Release-Namen der Debian-Version, die Sie verwenden möchten. Aktuell ist Bookworm die stabile Debian-Version.

Das Script benötigt einige Zeit, um automatisch eine Debian-Installation einzurichten. Sie sehen unter anderem den debootstrap-Prozess, die Locale-Generation, Download und Installation von Software-Paketen von den Debian-Servern, Generieren von neuen SSH-Keys und Server-Zeit. Möglicherweise erscheinen einige Fehlermeldungen wegen fehlenden optionalen Bestandteilen. Dies ist aber kein Problem, solange die folgenden Befehle funktionieren.

- Starten Sie nun den erstellten Container:

```
sudo lxc-start -n test
```

- LXC bietet auch gleich einen einfachen Weg, um sich auf dem Container-Betriebssystem einzuloggen:

```
sudo lxc-attach -n test
```

Wenn Sie nun die vorhandenen Dateien und Verzeichnisse anschauen, finden Sie eine mehr oder weniger komplette Betriebssystem-Umgebung vor. Darin können auch weitere Programme installiert werden.

5.2. Sicherheitsprobleme mit «privileged» Containern

Container, die mit dem root-User erstellt und gestartet werden, werden «privileged» genannt. Diese haben grundsätzlich auch auf dem Host-Betriebssystem root-Rechte, nicht nur innerhalb des Containers. Zwar sind die Isolations-Mechanismen vorhanden, vollständig sicher sind privileged Container allerdings nie. In älteren Versionen von LXC konnte beispielsweise trivial aus dem Container heraus das Host-Betriebssystem neu gestartet werden.

Sicherheitslücken wie CVE-2019-5736 betreffen dann auch nicht nur LXC, sondern alle ähnlichen Containerlösungen; runC und somit Docker, Kubernetes, cri-o und Containerd sowie Apache Mesos weisen die gleiche Schwachstelle auf. LXC stellt sich inzwischen auf den Standpunkt, dass **privileged Container nie sicher sein werden** und man diese generell nicht einsetzen sollte:

> "LXC upstream's position is that those containers aren't and cannot be root-safe.
>
> We are aware of a number of exploits which will let you escape such containers and get full root privileges on the host. Some of those exploits can be trivially blocked and so we do update our different policies once made aware of them. Some others aren't blockable as they would require blocking so many core features that the average container would become completely unusable."

In der Praxis ist das Problem, dass sowohl LXC als auch Docker und andere Tools es sehr einfach machen, unsichere Container einzurichten und zu betreiben; und sichere «unprivileged»-Container nicht nur viel schwieriger zu konfigurieren sind, sondern auch die entsprechende Dokumentation mangelhaft ist.

5.3. root-Rechte ausserhalb des Containers

- Zur Veranschaulichung können Sie eine Datei im Wurzelverzeichnis innerhalb des Containers erstellen:

```
echo asdf > test.txt
```

- Verlassen Sie nun den erstellten Container (mit exit oder Ctrl-D)
- und stoppen Sie diesen:

```
sudo lxc-stop -n test
```

Die gerade erstellte Textdatei befindet sich aus Sicht des Host-Betriebssystems im Verzeichnis /var/lib/lxc/test/rootfs/.

- Lassen Sie sich die Dateiattribute mit ls anzeigen:

```
sudo ls -l /var/lib/lxc/test/rootfs/test.txt
```

Wie Sie sehen, gehört die Datei auch ausserhalb des Containers dem root-User.

Dies ist beispielsweise problematisch, wenn Sie Dateien zwischen Host und Container austauschen möchten und dazu wie in der Praxis üblich einen Teil des Host-Dateisystems zusätzlich im Container einbinden (sogenannter bind-mount).

Es sind bei privileged Containern also **Benutzer nicht komplett isoliert**. Die **Dateisystem-Isolation funktioniert aber immer noch**, die Datei wurde im Container erstellt, und aus dem Container heraus kann man nicht auf Dateien ausserhalb des Containers zugreifen.

- Schliesslich löschen Sie den Container wieder:

```
sudo lxc-destroy -n test
```

5.4. Unprivileged Container mit LXC

Nun soll ein sicherer «unprivileged» Container erstellt werden. Wie erwähnt ist die Konfiguration viel aufwändiger und die Dokumentation weniger gut – deshalb entstehen um Container-Lösungen auch weitere Abstraktionsschichten wie LXD oder libvirt. *Was passiert, wenn Sie den* lxc-create*-Befehl oben ohne* sudo *laufen lassen?*

Vollständiges Setup und Konfiguration eines unprivileged Containers würde den Rahmen dieses Laborversuchs sprengen. Stattdessen gehen wir den einfachen Weg aus der LXC-Dokumentation und laden ein fertig konfiguriertes Image aus dem Internet. Dies führt zu neuen sicherheitsmässig problematischen Aspekten – Sie vertrauen darauf, dass das Image korrekt konfiguriert ist und dass der Ausliefer-Server nicht von Angreifern kontrolliert wird. Falls Sie produktiv mit Containern arbeiten, sollten Sie natürlich ein eigenes Template für das Setup von unprivileged Containern ausarbeiten.

Eine minimale Konfiguration für den Container befindet sich bereits in ~/.config/lxc/default.conf – beispielsweise wurde das AppArmor-Sicherheitsfeature von Debian ausgeschaltet und der Container erhält keinen Netzwerk-Adapter. Eine stabile Lösung für AppArmor mit unprivileged Containern ist zum aktuellen Zeitpunkt noch in der Entwicklungsphase.

Das Download-Template enthält nun die Möglichkeit, aus einer Reihe von verschiedenen Containern auf dem Downloadserver auszuwählen.

- Sie können die möglichen Containertypen auflisten:

```
lxc-create -n test -t download
```

- Brechen Sie den Vorgang mit **Ctrl-C** wieder ab.
- Löschen Sie den abgebrochenen Container wieder:

```
lxc-destroy -n test
```

- Erstellen Sie nun einen unprivileged Container mittels Download-Template und diesen Parametern:

```
lxc-create -n test -t download -- -d debian -r bookworm -a amd64
```

Wie Sie sehen, brauchen Sie dafür keine sudo-Rechte. Zusätzlich zur Debian-Version geben Sie nun auch das Betriebssystem (Debian) und die CPU-Architektur (amd64) an. Mit zusätzlichem Aufwand können Sie so z.B. auch einen Ubuntu-Container oder einen Container für eine andere CPU-Architektur erstellen.

- Starten Sie den Container und öffnen Sie das Shell-Interface:

```
lxc-unpriv-start -n test
```

```
lxc-unpriv-attach -n test
```

Die Befehle sind nun anders! Mit dem Zusatz «unpriv» wird jeweils ein Script gestartet, welches die nötigen Aufgaben automatisiert, welche zum Start eines Containers ohne root-Rechte zusätzlich nötig sind.

- Sie bekommen erneut eine root-Shell, was Sie verifizieren können:

```
id
```

```
whoami
```

Allerdings sind einige Dinge auffällig; beispielsweise können Sie nicht ins eigene Benutzerverzeichnis wechseln (cd ohne Argumente wechselt ins Verzeichnis ~).

- Versuchen Sie, ins Benutzerverzeichnis zu wechseln:

```
cd
```

Was passiert beim Ausführen des cd-Befehls?

Offenbar ist das Benutzerverzeichnis ~ auf den labadmin-Benutzer gesetzt. Dieser existiert allerdings im Container gar nicht.

Sie bemerken also ein Problem mit dem LXC-Container: Offenbar werden Environment-Variablen übernommen oder falsch gesetzt.

- Dies kann einfach überprüft werden:

```
printenv USER
```

```
printenv HOME
```

Obwohl Sie innerhalb des Containers als root-User agieren, sind die Environment-Variablen $USER und $HOME immer noch für den labadmin-User gesetzt. Dies ist wichtig zu wissen, um Informations-Leaks zu vermeiden. Starten Sie deshalb lxc-attach immer mit dem Parameter –clear-env.

- Verlassen Sie den Container wieder und versuchen Sie es mit diesem Parameter nochmals:

```
exit
```

```
lxc-unpriv-attach -n test --clear-env
```

Was ist die Ausgabe von printenv USER *innerhalb des Containers ohne* –clear-env*? Was passiert mit* –clear-env*?*

Es bleibt herauszufinden, ob das Problem mit root-Rechten auf Dateien ausserhalb des Containers mit dem unprivileged Container gelöst ist.

- Erstellen Sie nochmals eine Textdatei innerhalb des Containers

```
echo asdf > /test2.txt
```

- Verlassen Sie den Container wieder
- und schauen Sie nach, wie die entsprechende Datei auf dem Host aussieht:

```
ls -l .local/share/lxc/test/rootfs/test2.txt
```

Was wird für diese Datei als Besitzer und Gruppe angezeigt?

Die Datei gehört offenbar weder root noch labadmin, sondern einem unbekannten Benutzer mit sechsstelliger numerischer user-ID. Somit hat der **Unprivileged Container eine funktionierende Benutzer-Isolation**, und der root-User im Container kann keine Dateien auf dem Host mit root-Rechten bearbeiten.

- Stoppen und löschen Sie den Container nun wieder:

```
lxc-stop -n test
```

```
lxc-destroy -n test
```

6. Virtuelle Maschinen

Anstatt gewisse Teile eines Betriebssystems zu isolieren, gibt es auch die Möglichkeit, einfach ein ganzes Betriebssystem in einer virtuellen Maschine zu starten. Ursprünglich wurde dazu auf dem Host-System eine CPU in Software emuliert, inzwischen haben Prozessoren auch Hardware-Unterstützung für VMs.

Während des Studiums hatten Sie schon öfters Kontakt zu VM-Software wie VirtualBox oder VMWare. Unter GNU/Linux existiert der Emulator QEMU, welcher die Hardware eines PCs simuliert. Dieser verwendet KVM, ein Linux-Kernel-Modul für Hardware-Virtualisierung. Mit QEMU lassen sich beispielsweise auch diverse alternative CPU-Architekturen simulieren.

In einer emulierten Maschine wird ein vollständiges Betriebssystem inklusive Kernel auf einer virtuellen Festplatte installiert. Aus Sicht des virtualisieren Betriebssystems («Guest») macht es idealerweise keinen Unterschied, ob es direkt auf echter Hardware läuft oder nur auf der emulierten CPU.

Im Vergleich mit Containern ist hier also **auch der Kernel isoliert**, und generell wird nichts geteilt zwischen Host und Guest, ausser natürlich der Hardware.

Installation eines Betriebssystems mit gewünschtem Sicherheitsniveau ist nicht trivial, deshalb verwenden Sie auch hier ein bereits fertig installiertes Debian.

6.1. Copy-on-Write

Häufig werden auf einem Host mehrere VMs gestartet, um für verschiedene Anwendungen jeweils eine eigene Isolierte Umgebung zu haben. Diese bauen aber alle auf dem gleichen Basis-Betriebssystem auf, so dass viel Speicherplatz verschwendet wird, wenn für jede VM eine neue virtuelle Festplatte mit grösstenteils gleichem Inhalt verwendet wird.

Dieses Problem wird mit Copy-on-Write-Formaten gelöst: Sie erstellen eine neue Festplatte auf Basis einer existierenden Installation. In der neuen virtuellen Festplatte werden nur Änderungen gegenüber dem Original gespeichert. Dieses Prinzip nennt man Copy-on-Write. Als Nebeneffekt können Sie von vorne beginnen, wenn etwas schiefläuft, ohne jedesmal den Betriebssystem-Installationsprozess zu wiederholen.

Für QEMU bietet sich das qcow2-Format an.

- Erstellen Sie eine virtuelle qcow2-Festplatte auf Basis des existierenden Debian-Images:
  (Achtung: muss alles auf eine Zeile!)

```
qemu-img create -f qcow2 -F qcow2 -o backing_file=~/isolation/debian10-amd64
-422.qcow2 test.qcow2
```

Wie gross ist das originale Debian-Image? Wie gross ist das neu erstellte test-Image? Verwenden Sie ls *mit den Flags* -lh*.*

6.2. QEMU starten

- Starten Sie nun QEMU mit dem neu erstellten Image:
  (Achtung: muss alles auf eine Zeile!)

```
sudo qemu-system-x86_64 -enable-kvm -cpu host,pmu=off -m 512M -smp 1 -drive \
file=test.qcow2,format=qcow2 -display none -serial stdio
```

Der Schritt «Loading initial ramdisk» kann hier einige Zeit dauern.

Wir emulieren also ein x86_64-System (bzw. amd64). Das KVM-Kernel-Modul wird verwendet und die emulierte CPU soll die gleiche sein wie die des Host-Betriebssystems. PMU, ein Zusatz-Feature, wird dabei ausgeschaltet. Die Maschine erhält 512MB an RAM und einen CPU-Kern. Als Fesplatte wird das test-Image verwendet und ein Display ist nicht angeschlossen. Um dennoch mit dem VM-Betriebssystem interagieren zu können, soll auf dem Terminal eine serielle Konsole simuliert werden.

Um eine virtuelle Maschine mit QEMU zu starten, sind sudo-Rechte notwendig. Anders als bei Containern hat dies aber keinen negativen Einfluss auf das Host-Betriebssystem, da nichts zwischen Host und VM geteilt wird.

Das Dateisystem der VM ist in einer virtuellen Festplatte und unabhängig vom Host-Dateisystem; Benutzer- und Prozessmanagement ist innerhalb der VM komplett separat – wie bei einem normalen Betriebssystem. Den gleichen Test wie mit dem Container können wir also nicht wiederholen – wenn Sie innerhalb der VM eine Datei erstellen, taucht diese nirgends auf dem Host auf.

6.3. Zerstörbare Umgebung

Nach etwas Warten sollte die Debian-VM gebootet haben. Sie können sich direkt als root einloggen ohne Passwort – dies sollte für den Produktiveinsatz natürlich geändert werden.

- Login mit Benutzer **root**, Passwort leer lassen

Vielleicht haben Sie schon einmal vom berüchtigten rm -rf / gehört – der Befehl, der sämtliche Dateien löscht. Offenbar wurde die Möglichkeit des versehentlichen Zerstörens eines gesamten Betriebssystems genügend oft unbeabsichtigt eingesetzt, dass der rm-Befehl inzwischen einen zusätzlichen Sicherheits-Parameter verlangt.

Innerhalb der VM sollte es aber möglich sein auszuprobieren, was genau passiert, ohne auf dem Host-Betriebssystem irgendwas kaputt zu machen.

- **Überprüfen Sie vorher nochmals, dass sie sich beim Eintippen des Befehls tatsächlich innerhalb der QEMU-VM befinden!**
- Probieren Sie in der VM den Befehl aus:

```
root@debian:~# rm -rf /
```

Wie erwähnt weigert sich rm und verlangt zusätzlich das Flag.

- Versuchen Sie es nochmals, diesmal mit dem Extra-Flag.

```
root@debian:~# rm -rf / --no-preserve-root
```

Nun müssen Sie einige Zeit warten, in der rm alle Verzeichnisse und Dateien durchgeht. **Dabei erhalten Sie tausende Zeilen von Fehlermeldungen** für die Dateien in den Pseudo-Dateisystemen /proc und /sys.

Dies kann nun einige Minuten dauern, etwas Geduld ist gefragt. Am Ende erhalten Sie möglicherweise noch eine Meldung zum Absturz des dbus-daemon, der nicht damit umgehen kann, dass eine Datei gelöscht wurde.

Jetzt versuchen Sie herauszufinden, welche Daten noch vorhanden sind und wie das System sich verhält.

Was passiert, wenn Sie sich mit* ls *die Dateien im Wurzelverzeichnis anzeigen lassen?*

Die VM ist nun nicht mehr sehr nützlich.

- Da kein shutdown-Befehl mehr existiert, können Sie die VM mithilfe des /proc/-Interfaces herunterfahren:

```
echo 1 > /proc/sys/kernel/sysrq
```

```
echo o > /proc/sysrq-trigger
```

Das ist ein **kleines o und kein 0** in der zweiten Zeile.

Der echo-Befehl funktioniert noch, weil dieser in bash eingebaut ist, und bash wird ja als unsere Shell weiterhin ausgeführt und befindet sich somit noch im RAM der VM.

Sie befinden sich nun wieder ausserhalb der VM.

- Versuchen Sie nun, die VM nochmals zu starten mit dem gleichen qemu-Befehl wie zuvor.

Was passiert beim Start der VM?

Sie können mit **Ctrl-C** qemu wieder beenden.

- Löschen Sie die erstellte VM wieder:

```
rm test.qcow2
```

7. Verständnisfragen

Welche der folgenden Dinge sind isoliert: Dateisystem, Netzwerk, Kernel, Benutzer, Prozesse?

	chroot	Container (privileged)	Container (unprivileged)	VM
Dateisystem				
Netzwerk				
Kernel				
Benutzer				
Prozesse				

Lässt sich eine isolierte Umgebung für den Mechanismus im Kontext eines normalen Benutzers verwenden? Lässt sich diese ohne sudo starten?

Description	chroot	Container (privileged)	Container (unprivileged)	VM
Start ohne root-Rechte möglich, also kein sudo nötig				
Start nur mit root-Rechten, aber läuft später als nicht-root (privilege drop)				
Läuft nur mit root-Rechten				

Ordnen Sie die Mechanismen chroot, Container und VM nach Festplattenspeicher-Verbrauch.

Zusatzfrage: Gegen welche Klasse von Exploits hilft keine der drei vorgestellten Mechanismen? Gibt es eine Möglichkeit, verschiedene Dienste auch bei solchen Exploits zu isolieren?

8. Was man aus der heutigen Lektion mitnehmen sollte

Sie haben gesehen, welche verschiedenen Stufen der Isolation in modernen Betriebssystemen existieren und diese ausprobiert. Für verschiedene Anwendungsfälle sind verschiedene Lösungen besser geeignet. Mehr Sicherheit ist oft mit zusätzlichem Aufwand verbunden oder kostet extra Speicher, RAM und CPU-Zeit.

Bevor ein Tool zur Isolation eingesetzt wird, sollte man wissen, was dieses im Hintergrund macht und sich über dessen Schwächen und Einschränkungen informiert haben.

8.1. Notizen

Notizen