Simulierte Netze mit User Mode Linux als Basis für Firewall-Tests

Schein-Netz

Ob die Planung eines neuen Netzes realistisch und beispielsweise die Firewall wirksam konfiguriert ist, zeigt meist erst die Praxis. Das Vabanquespiel am lebenden Objekt und unter Zeitdruck ist aber nicht jedermanns Sache: Mit User Mode Linux simuliert schon ein einzelner Rechner realitätsnah ganze Netze. Christian Hammers

Wer ein Linux-System als Router und Firewall installieren und dem Kunden fertig konfiguriert zuschicken muss, sollte die Einstellungen vorher testen. Sonst heißt es hinfahren und nachkonfigurieren. Für den Test ist es nicht nötig, dutzende PCs, Switches und Router zu beschaffen: Eine Simulation genügt meist. Dank UML (User Mode Linux) laufen auf einem einzelnen Rechner gleichzeitig mehrere virtuell vernetzte Linux-Systeme. Als Beispielumgebung dient im Folgenden ein Netz mit einer Firewall, einer DMZ und einem Intranet-Host (siehe Abbildung 1).

Das UML-Projekt wurde 1999 von Jeff Dike ins Leben gerufen. Die Projekt-Homepage [1] enthält unter anderem das sehr lesenswerte Howto sowie das Archiv der Mailingliste und etliche kurze Tutorials zu verschiedenen Aspekten. Der Name deutet es schon an: UML nutzt einen modifizierten Linux-Kernel, den ein normaler User wie ein gewöhnliches Programm startet. Dieser zusätzliche Kernel ist die Hülle für ein weiteres Linux-System, das in der simulierten Umgebung läuft. Eine unter UML ausgeführte Applikation bemerkt nicht, dass ihr Kernel selbst als Userspace-Applikation auf einem weiteren Kernel aufsetzt.

Virtuelle Architektur

Um dieses Stapeln von Linux-Kerneln zu ermöglichen, erhält der Kernel eine weitere Architektur, die alle Hardware-nahen Systemaufrufe abfängt. Während bestehende Architekturen – etwa i386, s390, m68k – diese Aufrufe an die Hardware weiterleiten, reicht UML sie zum Wirtssystem durch. Der Overhead ist dabei recht gering, jedenfalls deutlich niedriger als bei virtuellen Maschinen wie VMware, Bochs oder Plex 86.

Zum Starten eines UML-Kernels sind keine Root-Rechte nötig. Innerhalb der UML-Umgebung gilt die Rechteverwaltung des simulierten Systems, aber vom Wirtssystem aus hat der User, der UML gestartet hat, alle Rechte. Root-Rechte braucht nur, wer die Netzwerkkonfiguration des Hostsystems verändern muss, um die UML-Systeme zu verbinden. Die dazu nötigen Einstellungen kann der Administrator jedoch vorbereiten.

Ein handelsüblicher PC verkraftet problemlos ein halbes Dutzend virtueller Systeme. Für ein kleines UML-Testsystem genügen schon 128 MByte Plattenplatz und 48 MByte RAM. Voraussetzungen sind nur ein passendes Hostsystem, der ausführbare UML-Kernel und ein eigenes Dateisystem. Den Kernel gibt es auf der Projekt-Homepage oder in einigen Distributionen (beispielsweise Debian) als fertiges Paket.

Es wäre unklug, zwei Kernel gleichzeitig in ein gemeinsames Dateisystem schreiben zu lassen – solch unkoordiniertes Handeln führt zu Datensalat. Jedes UML-System braucht sein eigenes Filesystem. Wer keine zusätzlichen Partitionen anlegen will, kann jedes UML-System auch in ein eigenes Unterverzeichnis des Wirtssystems schreiben lassen (ähnlich »chroot«) oder jedem Gastsystem eine Datei spendieren, in der er das Filesystem anlegt (Loopback-Prinzip). Abbildung 2 zeigt die Varianten.

Filesystem des Wirts oder Loop-Device

Die Verzeichnisvariante ist zwar unkompliziert, UML muss dazu aber mit Root-Rechten laufen. Andernfalls dürfte es die Dateibesitzrechte im Wirtssystem nicht ändern. Eine Loopback-Datei, die das komplette Dateisystem enthält, umgeht dieses Problem. Für dieses Feature ist der Loopback-Device-Support im Kernel zuständig (siehe Abbildung 3). Das Loop-Modul bindet eine Datei an ein Blockdevice, das sich mit »mount« in das Dateisystem einhängen und wie eine Festplatte oder wie eine Partition beschreiben lässt.

Um eine Loopback-Datei vorzubereiten, ist zunächst ein leeres File der gewünschten Größe nötig. Das lässt sich leicht mit einem »dd«-Kommando anlegen (Zeile 1 in Listing 1). Die Programme »mkfs.ext3« und »fsck.ext3« greifen direkt auf die Loopback-Datei oder indirekt über das Blockdevice zu (Zeile 6) und legen das gewünschte Dateisystem an. Nun lässt sich die Loopback-Datei mounten (Zeile 13) und mit Inhalt füllen. Achtung: Nicht vergessen,

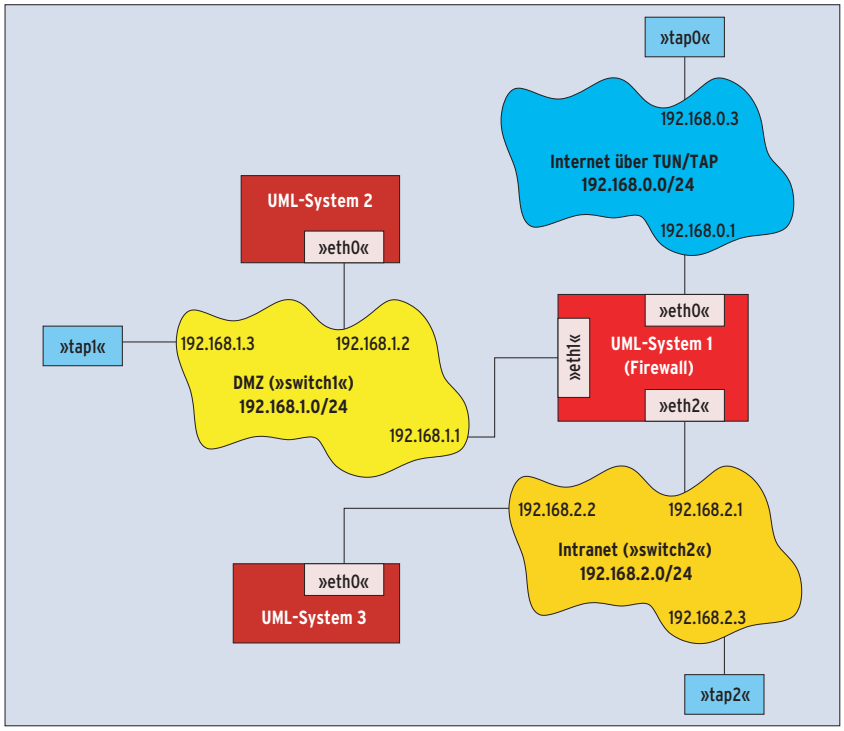


Abbildung 1: Dieses Testnetz mit drei UML-Systemen und drei Netzen wird komplett auf einem Rechner simuliert. Die drei TAP-Interfaces verbinden den realen Host mit der simulierten Welt.

die Datei nach dieser Aktion wieder zu unmounten (Listing 2), bevor sie vom UML-Kern verwendet wird.

Platz sparen

Muss ein Wirt viele Hosts simulieren, benötigt er auch viele Filesysteme und viel Plattenplatz. Häufig gleichen sich die Filesysteme aber in weiten Bereichen, nur Logfiles und einige Konfigurationsdateien sind unterschiedlich. Diesen Umstand nutzt das COW-Feature (Copy on Write) des UBD-Treibers (UML Block Device), um Plattenplatz einzusparen. Als Grundlage für ihr Dateisystem dient ein gemeinsames Backing-File, das sich – einmal angelegt – nicht mehr verändert. Alle Änderungen legt der UBD-Treiber in einer Host-spezifischen COW-Datei ab. Existiert dieses File beim Start nicht, legt er es automatisch an.

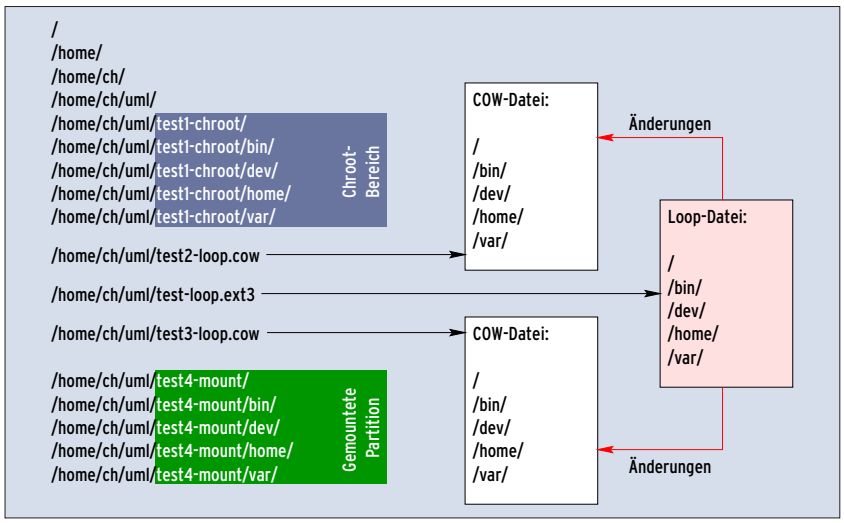


Abbildung 2: Jeder UML-Host braucht sein eigenes Dateisystem. Neben einem Chroot-Bereich (blau, Test 1) oder einer eigenen Partition (grün, Test 4) bieten sich Loop-Dateien an. Mit der platzsparenden COW-Technik speichert UML nur Änderungen in eigenen Files (weiß, Test 2 und 3) und nutzt eine gemeinsame Basis (pink).

Die COW-Datei ist äußerlich so groß wie die Basisdatei, enthält aber fast nur Nullen. Gängige Dateisysteme wie Ext 2 und Ext 3 erkennen sie als sparse (karg) und speichern sie platzsparend. Der Trick: Im Inode sind leere Blöcke (solche, die nur Null-Bytes enthalten) zwar verzeichnet, sie belegen aber keinen Plattenplatz. Den Unterschied zeigen »ls -l -s« oder »du -m *«. In der ersten Spalte steht der tatsächlich belegte Platz, er orientiert sich an der Blockgröße.

Die drei Systeme aus Abbildung 1 unterscheiden sich nur in ihrer Netzwerkkonfiguration, COW spart daher viel Platz. Es sind vier Files nötig: Auf der Backing-Datei »uml-image.ext3« wird zu Beginn das Betriebssystem installiert. Die UML-Systeme greifen nur auf die drei COW-Dateien zu: »uml1.cow«, »uml2.cow« und »uml3.cow«. Das Backing-File muss nun unverändert bleiben.

Linux in UML installieren

Ein UML-System lässt sich nicht wie gewohnt per CD-ROM installieren. Es wäre auch unpraktisch, bei mehreren simulierten Maschinen jedes Mal eine Distribution manuell einzuspielen. Für UML-Systeme gibt es bessere Wege, um das Zielsystem in die Loopback-Datei oder das UML-Verzeichnis zu installieren. Debian eignet sich sehr gut für diese Auf-

Listing 1: Loopback-Datei anlegen

```
01 # dd if=/dev/zero of=uml-image.ext3 bs=1M count=400
02 400+0 Records ein
03 400+0 Records aus
04 419430400 bytes transferred in 32,645106 seconds
   (12848186 bytes/sec)
05 # losetup /dev/loop0 uml-image.ext3
06 # mkfs.ext3 -j /dev/loop0
07 mke2fs 1.34-WIP (21-May-2003)
08 Dateisystem Label=
09 OS type: Linux
10 Blockgrösse=1024 (log=0)
11 Fragmentgrösse=1024 (log=0)
12 ...
13 # mount /dev/loop0 /mnt/uml-image
```

Listing 2: Loopback-Datei schließen

```
01 # umount /mnt/uml-image
02 # losetup -d /dev/loop0
03 # fsck.ext3 uml-image.ext3
04 e2fsck 1.34-WIP (21-May-2003)
05 uml-image.ext3: clean, 11/102400 files, 21167/409600
   blocks
```

gabe. Die Applikationen sind streng in mehrere Pakete für die wirklich notwendigen Dateien und optionale Bestandteile wie Dokumentationen oder Headerfiles aufgeteilt. Die Default-Einstellungen sind sehr restriktiv gehalten.

Debians Bootstrap-Tool »debootstrap« legt ein minimales Debian-System in einem Verzeichnis an. Mit Chroot oder bereits im laufenden UML kann man es beliebig um Pakete erweitern und konfigurieren. Das Programm erwartet die gewünschte Distribution, das Zielverzeichnis und einen Mirror als Parameter (siehe »/etc/apt/sources.list«). Da Debootstrap das System nicht interaktiv installiert, muss man das Ergebnis nachkonfigurieren. Für das Testnetz sind im Wesentlichen folgende Host-spezifischen Einstellungen erforderlich:

- Netzwerkkonfiguration in »/etc/network/interfaces« vornehmen.
- Hostname in »/etc/hosts« und »/etc/hostname« eintragen.
- Firewallskript schreiben und in »/etc/runlevel.conf« oder per RC-Skript in »/etc/rc*.d/« aktivieren.

Listing 3: Sparse-Effekt bei COW-Dateien

```
01 # ls -alhs *cow *ext3
02  25M -rw-r--r--   1 root   root   501M 2003-08-23 18:17 uml1.cow
03  25M -rw-r--r--   1 root   root   501M 2003-08-22 14:05 uml2.cow
04  32M -rw-r--r--   1 root   root   501M 2003-08-23 18:18 uml3.cow
05 353M -rw-r--r--   1 root   root   500M 2003-08-21 00:08 uml-image.ext3
```

Listing 4: Debootstrap installiert ein System

```
01 # cd /mnt/uml-image
02 # df -h .
03 Filesystem    Größe Benut  Verf Ben% Eingehängt auf
04 /dev/loop0     388M  8,1M  360M   3% /mnt/uml-image
05
06 # debootstrap --include=tcpdump,iproute woody . ftp://ftp.de.debian.org/debian
07 I: Retrieving ftp://ftp.de.debian.org/debian/dists/woody/Release
08 I: Validating /mnt/uml-image/./var/lib/apt/lists/debootstrap.invalid_dists_woody_Release
09 I: Retrieving ftp://ftp.de.debian.org/debian/dists/woody/main/binary-i386/Packages.gz
10 I: Validating /mnt/uml-image/./var/lib/apt/lists/debootstrap.invalid_dists_woody_main_binary-
   i386_Packages.gz
11 I: Retrieving ftp://ftp.de.debian.org/debian/pool/main/a/adduser/adduser_3.47_all.deb
12 I: Validating /mnt/uml-image/./var/cache/apt/archives/adduser_3.47_all.deb
13 [...]
14
15 # chroot .
16 # dpkg-reconfigure --priority low --all
17 # exit
18
19 # df -h .
20 Filesystem    Größe Benut  Verf Ben% Eingehängt auf
21 /dev/loop0     388M  102M  266M  28% /mnt/uml-image
```

Um keine bösen Überraschungen bei einem späteren »fsck« zu erleben, sollte auf den UML-Systemen der Device-Eintrag »/dev/ubd0« existieren. Bei Bedarf erzeugt »mknod /dev/ubd0 b 98 0« diese Gerätedatei.

UML starten

UML erfordert keine Veränderungen am Wirtssystem. Die UML-Kernel sind aus seiner Sicht gewöhnliche Userspace-Prozesse und lassen sich auch über ein normales Kommando starten: »linux«. Allerdings ist es ratsam, das SKAS-Patch (Separate Kernel Address Space) in den Wirtskernel einzuspielen, es steigert die Performance auf das Zwei- bis Vierfache. Da der UML-Kern aus dem normalen Linux-Quellcode entsteht, akzeptiert er auch die meisten Optionen, die Lilo oder Grub einem gewöhnlichen Kernel übergeben würden. Einige Optionen gelten aber nur für UML oder sind anders anzuwenden.

Die meisten Optionen verstehen mehrere Parameter, sie sind durch Kommas voneinander getrennt. Wer nicht alle Parameter angeben will, lässt die restlichen einfach weg. Da der Kernel die Bedeutung des Parameters aus seiner Position ermittelt, muss aber die Anzahl der Kommas korrekt sein. Daher kommt es häufig vor, dass mehrere Kommas unmittelbar nacheinander stehen, um einen später folgenden Parameter an die richtige Stelle zu setzen.

Eine Frage des Filesystems

Die wichtigste Option legt fest, auf welche Art der UML-Kern auf das Dateisystem zugreift. Er darf schließlich nicht unmittelbar auf die Festplatten zugreifen, sondern muss eine der beschriebenen Techniken nutzen (Loopback oder Verzeichnis). Die Optionen »ubd« und »root« sind hierfür zuständig. Um das Loopback-Dateisystem aus dem File »woody1.ext3« innerhalb des UML-Systems als Blockdevice »/dev/ubda« zur Verfügung zu stellen, muss man dem Kernel die Option »ubd0 = woody1.ext3« übergeben:

```
linux ubd0=woody1.ext3
```

Per Default bootet UML auch von diesem Device. Eine zweite Option »ubd1 = *Datei2*« würde dem UML-System Zugriff auf ein weiteres Dateisystem geben. Der UML-Kernel kann sogar die Dateisysteme innerhalb der Loopback-Files selbst anlegen. »fdisk /dev/ubdb« legt eine neue Partition an, »mkfs.ext3 -j /dev/ubdb1« formatiert sie.

Folgender UML-Start sorgt dafür, dass der UML-Kernel sein Root-Dateisystem direkt aus einem Verzeichnis des Wirtssystems liest, ohne den Umweg über eine Loopback-Datei zu nehmen:

```
linux root=/dev/root rootfstype=hostfs ➥
rootflags=/Pfad/zum/Chroot/
```

Wer umgekehrt aus einem laufenden UML-System auf das Dateisystem seines Wirtssystems zugreifen will, nutzt das Host-Filesystem. »mount none -t hostfs /mnt/hostfs« bindet das Root-Filesystem des Hosts in das Verzeichnis »/mnt/hostfs« des UML-Systems ein.

Um die UML-Instanzen besser unterscheiden zu können, ist es möglich, ihnen beim Start mit »umid = *Name*« einen Namen zu geben. Dieser Name ist nicht

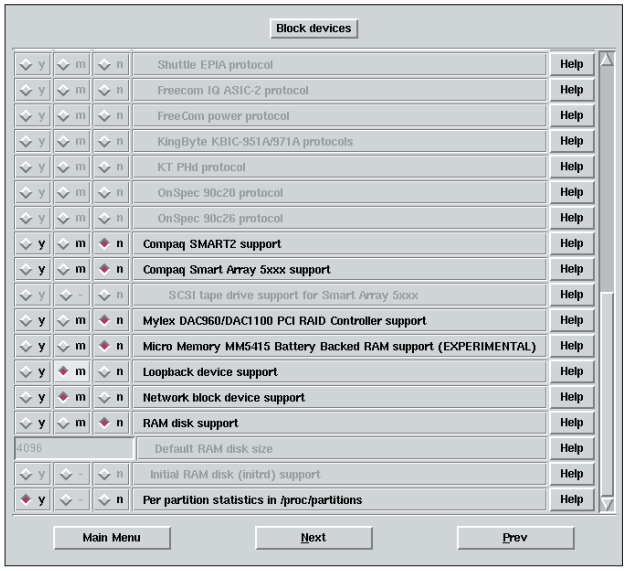


Abbildung 3: Ein Loopback-Device speichert ein komplettes Filesystem in einer normalen Datei. Um das Feature zu nutzen, muss im Kernel der »Loopback device support« als Modul aktivieren sein, »modprobe loop« lädt das Modul.

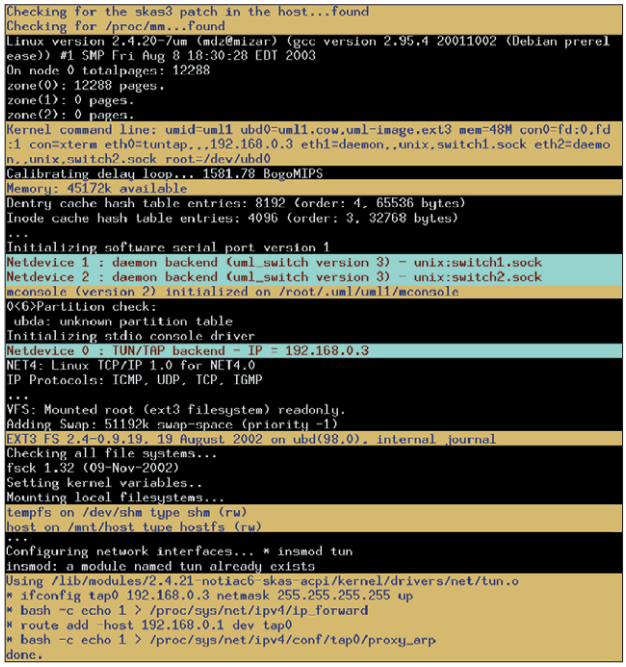


Abbildung 4: Wenn ein UML-System bootet, erscheinen die gewohnten Meldungen des Linux-Kernels. Die interessanten Abschnitte sind farbig hinterlegt (gekürzt).

mit dem Hostnamen des UML-Systems zu verwechseln, obwohl es durchaus sinnvoll ist, beide gleich zu halten. Der Name erscheint in der Prozessliste, außerdem gestaltet sich die Arbeit mit dem UML-Verwaltungstool »uml_mconsole« bequemer (siehe Kasten „UML-Managementkonsole“).

Die »mem«-Option legt die Größe des Hauptspeichers fest, den das UML nutzen darf. Auch bei einem normalen Linux-Kernel begrenzt diese Option den Speicher künstlich. Eine typische Größe wäre »mem = 48M« für einen einfachen Router, auf dem kaum Prozesse laufen. Wer allerdings ein Diskless-System ohne Swap simulieren will, sollte an dieser Stelle nicht sparen und lieber 128 MByte oder mehr zuweisen.

Die zusätzlichen Konsolen

Per Default öffnen sich – je nach Distribution – beim Start von UML sechs XTerms. Über sie hat der User Zugang zu den (virtuellen) Konsolen des UML-Systems. Wem das zu viel ist, weil er sich nur per SSH in das simulierte System einloggen will, verringert die Anzahl der Getty-Prozesse in »/etc/inittab« (sie öffnen die Terminals) oder benutzt eine der anderen Terminal-Umleitungen. Besonders im X11-losen Betrieb ist es sinnvoll, die Bootmeldungen in einer Datei zu protokollieren und die Terminals (TTYs) zu den Pseudo-Terminals »/dev/pts/*« des Hostsystems umzuleiten:

```
linux … con0=fd:0,fd:1 con=pts | tee uml.log
```

Die »con0«-Option betrifft nur Konsole »0«, während das allgemeinere »con« für alle anderen gilt. Auf die »pts«-Devices kann man dann zum Beispiel mit »screen /dev/pts/9« zugreifen. Die entsprechende Nummer wird, falls beim

UML-Managementkonsole

UML unterstützt mehrere Techniken, um das simulierte Linux-System vom Wirtssystem aus zu administrieren. Der User kann direkt auf die Konsolen des UML-Systems zugreifen, sich per SSH in das System einloggen oder die »uml_mconsole« nutzen. Dieses Programm ersetzt unter anderem den physischen Zugang zur simulierten Rechnerhardware. So kann der Admin bei Abstürzen das System neu booten oder nachträglich Netzwerkgeräte und Festplatten hinzufügen oder entfernen.

Eindeutiger Name für jedes UML-System

Der Aufruf »uml_mconsole ~/.uml/*Name*/mconsole« startet die Konsole, der Name entspricht der beim Starten des UML übergebenen »umid«. Wer keinen Namen angegeben hat, muss hier mit einer zufälligen Zeichenkette rechnen. Wie diese Zeichenkette heißt, zeigt UML beim Booten in folgender Zeile:

```
mconsole (version 2) initialized on ↵
/home/ch/.uml/CrEc3S/mconsole
```

Neben den Resetknopf-ähnlichen Befehlen »halt« und »restart« gibt es zum sauberen Neustarten auch »cad«. Das Kommando simuliert die Tastenkombination [Ctrl]+[Alt]+[Del]. Die Konsole kennt mit »sysrq« auch die Sys-Rq-Befehle; im UML-Kernel müssen sie dazu unter »Kernel Hacking« aktiviert sein.

Außerhalb des UML reagieren Sys-Rq-Befehle auf die Kombination von [AltGr]+[SysRq] mit einer weiteren Taste. [SysRq] ist unterhalb von [Druck] zu finden; die Taste heißt auf einigen Tastaturen auch [S-Abf]. Die wichtigsten Tasten sind: [S] synchronisiert die gemounteten Filesysteme, [U] führt einen Readonly-Remount aller Platten durch, [B] bootet neu (ohne vorherigen Sync) und [H] gibt eine kurze Hilfe auf der Konsole aus. Eine genauere Beschreibung findet sich in den Kernelsourcen in der Datei »Documentation/sysrq.txt«.

Im laufenden Betrieb Hardware ergänzen

Zusätzliche Hardware verwaltet die UML-Managementkonsole mit »config *Device=Konfiguration*« und »remove *Device*«. Das funktioniert mit den UBD- und den Netzwerk-Treibern. Zum Beispiel setzt »config eth0=mcast« die erste Ethernet-Schnittstelle auf Multicast-Betrieb. Bei einigen Geräten ermittelt »config *Device*« den aktuellen Parameterstring. »remove *Device*« entfernt erwartungsgemäß das Gerät aus dem UML-System.

UML-Boot nicht bereits festgelegt, im Bootscreen angezeigt: »Virtual console 1 assigned device '/dev/pts/8'«.

Listing 5 zeigt einen kompletten Satz von Optionen am Beispiel des Startskripts für den Firewallhost aus dem anfangs beschriebenen Testnetz (Abbildung 1). Das UML-System erhält 48 MByte Speicher und trägt den Namen »uml1«. Sein Root-Filesystem entstammt der COW-Datei »uml1.cow« und dem Backing-File »uml-image.ext3«. Konsole »0« ist über die Stdin und Stdout (Filedeskriptoren» 0« und »1«) erreichbar, für die restlichen Konsolen öffnet sich je ein XTerm. Das erste Ethernet-Interface »eth0« ist per TUN/TAP-Modul an die IP-Adresse 192.168.0.3 angeschlossen. Achtung: Diese Adresse gilt für den Endpunkt im Wirtshost und nicht im UML-System. Die beiden anderen Interfaces verwenden den UML-Switch.

Netzwerke mit UML

UML vernetzt die simulierten Systeme und das Netzwerk des Hosts mit mehreren Techniken (siehe Abbildung 5):

- TUN/TAP verbindet UML-Systeme und ihren Wirts-Host.
- Multicast und UML-Switch verbinden UMLs miteinander.
- Weitere Netzwerktechniken sind im offiziellen Howto beschrieben.

Bei allen Methoden kann der Benutzer die MAC-Adressen der simulierten Interfaces selbst festlegen, meist ist das aber nicht erforderlich. Der UML-Kernel generiert sie gegebenenfalls automatisch aus der IP-Adresse, die dem Device als erste zugewiesen wird.

Listing 5: UML-Startskript

```
01 linux mem=48M umid=uml1 \
02     ubd0=uml1.cow,uml-image.ext3 \
03     con0=fd:0,fd:1 con=xterm \
04     eth0=tuntap,,,192.168.0.3 \
05     eth1=daemon,,unix,switch1.sock \
06     eth2=daemon,,unix,switch2.sock
```

Listing 6: UML-Switch

```
01 host# tunctl -t tap2
02 host# ip link set tap2 up
03 host# uml_switch -hub -tap tap2 -unix switch2.sock
04         Set 'tap2' persistent and owned by uid 0
05         uml_switch will be a hub instead of a switch
06         uml_switch attached to unix socket
   'switch2.sock'
07         New connection
08         New connection
09         Addr: fe:fd:c0:a8:02:02 New port 7
10         Addr: fe:fd:c0:a8:02:01 New port 6
```

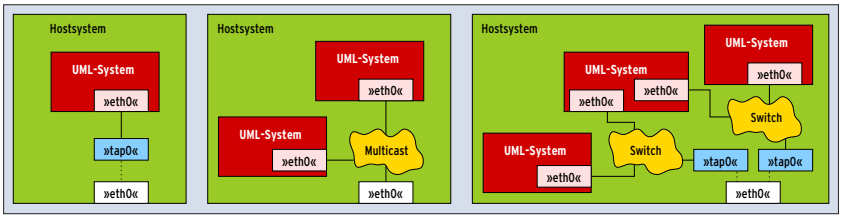


Abbildung 5: Die drei wichtigsten Techniken, um UML-Systeme zu vernetzen: Das TAP-Interface (links) verbindet lediglich Host und UML-System. Die Multicast-Methode vernetzt zwar UML-Systeme miteinander, hat aber Schwächen. Die universelle Variante benutzt den UML-Switch (rechts).

TUN/TAP

Damit das UML-System auch extern erreichbar ist, muss eins seiner simulierten Interfaces mit dem tatsächlichen Interface des Hostsystems verbunden sein. Das gelingt mit dem TAP-Treiber aus dem TUN/TAP-Kernelmodul: Er leitet Ethernet-Frames von einem Netzinterface an ein Programm – hier UML – weiter. UML lädt das Kernelmodul automatisch. Sollte es nicht verfügbar sein, kann man es in der Kernelkonfiguration im Menü »Network device support« unter »Universal TUN/TAP device driver support« aktivieren (Abbildung 6).

Ein Transfernetz verbindet das TAP-Interface und das entsprechende Ethernet-Interface des UML-Systems. Beide Seiten erhalten dazu eine IP-Adresse aus einem gemeinsamen Subnetz:

```
host# linux ... eth0=tuntap,,,192.168.0.3
uml1# ifconfig eth0 192.168.0.1 netmask ➥
      255.255.255.0 up
```

Um Pakete nach außen zu senden, verwendet UML die IP-Adresse des Wirtssystems als Default-Gateway. Der Wirt muss den Paketen aber eine neue Absenderadresse verpassen, damit die Antwortpakete auch ihren Weg zurück finden. Das geschieht über einen Source-NAT-Eintrag (Masquerading):

```
host# iptables -t nat -s 192.168.0.1 ➥
      -j SNAT --to-source Reale-Host-IP
uml1# route add default gw 192.168.0.3
```

Bei Testsystemen ist eine Verbindung mit dem Internet oft unnötig oder sogar gefährlich. Um dennoch einzelne Netzdienste nutzen zu können, lässt sich das Hostsystem über die »tuntap«-Technik auch ohne Masquerading als DNS- oder HTTP-Proxy benutzten. Bei Debian-Systemen ist der Apt-Proxy-Daemon besonders praktisch. Er ähnelt einem Webproxy, ist allerdings speziell für das Paketverwaltungsprogramm »apt-get« gedacht. Beim Update mehrerer UML-Systeme derselben Version spart der Proxy viel Zeit, ab dem zweiten UML sind keine neuen Downloads nötig.

Multicast

Eine trickreiche Methode, Pakete zwischen mehreren UML-Systemen auszutauschen, ist der Multicast-Transport. Er simuliert einen Hub: Alle Interfaces, die über dieselbe Multicast-Adresse angeschlossen sind, verhalten sich so, als wären sie mit einem gemeinsamen Hub verbunden. Beim Start der UMLs ist dazu für alle gewünschten Interfaces die Option »eth*N*=mcast« nötig sowie eventuell die Multicast-Adresse.

Innerhalb der UML-Systeme dürfen die Interfaces beliebige IP-Adressen erhalten. Der Vorgabewert für die Multicast-Adresse ist 239.192.168.1, laut RFC 2365 [2] liegt er in einem Bereich, der für die freie Benutzung innerhalb einer Organisation reserviert ist. Voraussetzung für diese Methode ist, dass Multicast im Kernel des Hostsystems aktiviert ist – üblicherweise ist das der Fall. Außerdem muss eine Multicast-fähige Netzwerkkarte (zum Beispiel eine Ethernetkarte) eingebaut sein.

Diese Netzwerkkarte ist der größte Haken bei dieser Methode: Sie sendet alle Multicast-Pakete auch nach außen. Die Daten kommen zwar nicht sehr weit, weil UML die Time to Live (TTL) auf »1« setzt – beim nächsten Router ist Schluss. Das macht diese Methode aber unsicherer und uneleganter als die im Folgenden

beschriebene Switch-Variante. Mit der Multicast-Methode ist zudem keine Verbindung zum Hostsystem möglich.

UML-Switch

Die bessere Alternative ist der Daemon »uml_switch«. Er simuliert einen Switch oder mit der Option »-hub« auch einen Hub. An diesen simulierten Netzknoten kann der Admin mehrere UMLs und das Hostsystem anschließen. Der Switch-Daemon läuft komplett im User Mode. Die Pakete müssen daher vom Kernel zum Daemon und dann wieder zurück kopiert werden. Unter dem Kopieren leidet zwar die Performance, für Tests ist das in der Regel aber kein Problem.
Die Switches lassen sich mit folgendem Kommando starten: »uml_switch -unix */Pfad*/switch1.sock«. Der Dateiname muss für jeden Switch einmalig sein. Er bezeichnet den Socket, zu dem sich die UMLs beim Start verbinden:

```
linux eth0=daemon,,unix,/Pfad/switch1.sock ➥
eth1=daemon,00:00:00:00:01:01,unix,➥
/Pfad/switch2.sock
```

Um neben den UMLs auch das lokale System anzuschließen, muss man dort ein TAP-Interface konfigurieren und es dem Switch beim Start mitteilen (siehe Listing 6). Auf diesem Weg lässt sich der Netzwerkverkehr auch vom Wirtssystem aus beobachten: Es genügt, den UML-Switch als Hub zu starten, alle Pakete sind dann auch über das TAP-Interface zu sehen. Das TAP-Interface benötigt dazu keine IP-Adresse und sollte auch keine erhalten. Der Kernel würde sonst davon ausgehen, dass er Pakete für das jeweilige Netz über dieses Interface schicken soll.

Brückenarbeit

Wer mehr Performance oder ausgefallenere Funktionen benötigt und ein komplizierteres Setup [4] nicht scheut, kann mit der Linux Virtual Bridge [3] eine Kernelspace-Variante des UML-Switch aufbauen. Ein Kernelmodul stellt hierbei ein Bridge-Interface zur Verfügung. An dies lassen sich dann beliebig viele TAP-Devices heften.
Für den Test der Konfiguration ist es sinnvoll, den Wirts-Host über eine TUN/TAP-Verbindung direkt an die simulierte Firewall anzuschließen. Er dient als Default-Gateway der Firewall und öffnet ihr den Weg nach draußen. Zwei Switches stellen die Verbindung zu den Hosts in der DMZ und zum Intranet her. Um den Datenverkehr auf den Switches zu analysieren, sind beide über je ein TUN/TAP-Interface angezapft (Abbildung 1):

```
host# ip route add 192.168.1.0/24 via ➥
      192.168.0.1
host# ip route add 192.168.2.0/24 via ➥
      192.168.0.1
```

Dieses Routing sagt dem Hostsystem, wie es die Netze in der DMZ und im Intranet erreicht. Danach kommunizieren die beiden Systeme in der DMZ und im Intranet auch mit dem Wirt (über IP 192.168.0.3). Beide Netze sind über das TAP-Interface zur Firewall erreichbar.

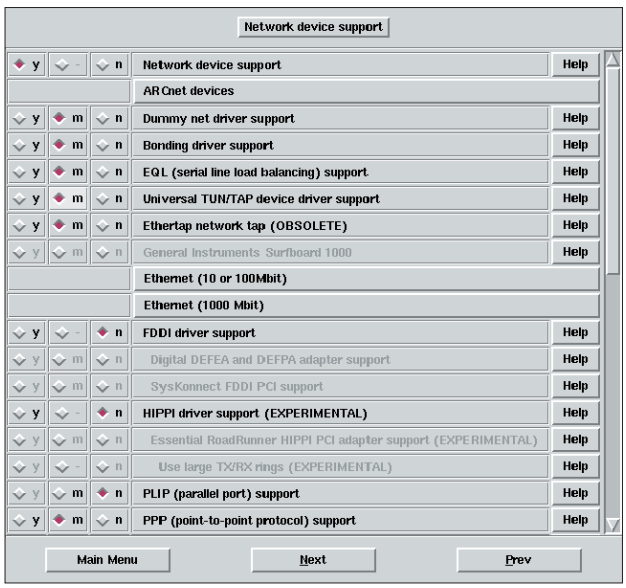
Mit diesen simulierten Netzen und Hosts ist das Testen einfacher als in einer realen Umgebung. Selbst bei gröbsten Fehlern in der Konfiguration hat der Tester immer noch direkten Zugang zu allen Systemen, er kann sich nie aussperren. Mit Tcpdump oder Ethereal schneidet er den Datenverkehr der einzelnen virtuellen Switches mit und prüft, an welcher Stelle es hakt, wenn Verbindungen nicht zustande kommen. Abbildung 7 zeigt einen Ping vom Intranet-Host über den virtuellen Switch 2 ins Internet, hier also das Hostsystem. Außerdem zeigt das Bild den Befehl, der zum Mitschneiden dieser Daten nötig ist.

Andere Anwendungszwecke

UML ist die ideale Basis für viele Tests. Zum Beispiel können Netzwerker mit der Zebra-Routing-Suite OSPF- und BGP-Routing erlernen und damit experimentieren oder neuere Technologien wie MPLS entwickeln. UML bietet sich auch für die Kernelentwicklung an: Ein Absturz ist nicht weiter tragisch und lässt sich sogar noch ganz gut debuggen. UML-Systeme kommen auch für so genannte Honeypot-Systeme zum Einsatz. Sie locken Netzwerkattacken an, um die Tools und Strategien der Cracker zu analysieren. *(fjl)* ■

Infos

[1] UML: [http://user-mode-linux.sf.net/]
[2] Administratively Scoped IP Multicast: [http://www.ietf.org/rfc/rfc2365.txt]
[3] Linux-Bridge: [http://bridge.sf.net/]
[4] Howto zur Integration von UML, virtual Bridges und Zebra: [http://www.lathspell.de/linux/uml/]




```
--(root@notiac)-(pts/8)-(17:39:36|23)--
--(~)# tcpdump -n -l -s1500 -i tap2
tcpdump: WARNING: tap2: no IPv4 address assigned
tcpdump: listening on tap2
17:39:40.469623 192.168.2.2 > 192.168.0.3: icmp: echo request (DF)
17:39:40.469865 192.168.0.3 > 192.168.2.2: icmp: echo reply
17:39:42.549615 192.168.2.2 > 192.168.0.3: icmp: echo request (DF)
17:39:42.549856 192.168.0.3 > 192.168.2.2: icmp: echo reply
17:39:44.629619 192.168.2.2 > 192.168.0.3: icmp: echo request (DF)
17:39:44.629870 192.168.0.3 > 192.168.2.2: icmp: echo reply

6 packets received by filter
0 packets dropped by kernel
```

▲ **Abbildung 7: Der Tcpdump-Mitschnitt auf dem virtuellen Switch 2 (über »tap2« angebunden) zeigt den Ping von 192.168.2.2 auf 192.168.0.3.**

◀ **Abbildung 6: Das TUN/TAP-Device stellt ein virtuelles Netzwerkdevice im Userspace zur Verfügung. In der Kernelkonfiguration ist der passende Eintrag »Universal TUN/TAP device driver support«.**