# Virtuelle Server unter Linux

Enrico Scholz

enrico.scholz@informatik.tu-chemnitz.de

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )
- sinnvoll: dedizierte Server
  - unterschiedliche Umgebungen (Libraries, Programme, Distributionen)
  - eigene Hostnamen & IPs
  - Abgrenzung der Zugriffsrechte

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )
- sinnvoll: dedizierte Server
  - unterschiedliche Umgebungen (Libraries, Programme, Distributionen)
  - eigene Hostnamen & IPs
  - Abgrenzung der Zugriffsrechte
- Oft: hohe Idle-Zeiten, aber low-end wegen Peaks nicht möglich

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )
- sinnvoll: dedizierte Server
  - unterschiedliche Umgebungen (Libraries, Programme, Distributionen)
  - eigene Hostnamen & IPs
  - Abgrenzung der Zugriffsrechte
- Oft: hohe Idle-Zeiten, aber low-end wegen Peaks nicht möglich
- Aber: Hardwarekosten, Raum, Kühlung, USV

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )
- sinnvoll: dedizierte Server
  - unterschiedliche Umgebungen (Libraries, Programme, Distributionen)
  - eigene Hostnamen & IPs
  - Abgrenzung der Zugriffsrechte
- Oft: hohe Idle-Zeiten, aber low-end wegen Peaks nicht möglich
- Aber: Hardwarekosten, Raum, Kühlung, USV

- Auch: Bereitstellung von root-Servern für Kunden

# Motivation



- Vielzahl von Diensten (FTP, HTTP, LDAP, KRB, DNS, . . . )
- sinnvoll: dedizierte Server
  - unterschiedliche Umgebungen (Libraries, Programme, Distributionen)
  - eigene Hostnamen & IPs
  - Abgrenzung der Zugriffsrechte
- Oft: hohe Idle-Zeiten, aber low-end wegen Peaks nicht möglich
- Aber: Hardwarekosten, Raum, Kühlung, USV

- Auch: Bereitstellung von root-Servern für Kunden

$\Rightarrow$ Optimal: mehrere dedizierte Server auf der selben Hardware

$\Rightarrow$ „virtuelle Server“

# Anforderungen



- Verhalten wie „normaler“ Server

# Anforderungen



- Verhalten wie „normaler“ Server
- Abgrenzung der Prozesse
  - `kill(2)`, `ptrace(2)`
  - /etc/init.d/sshd restart

# Anforderungen



- Verhalten wie „normaler“ Server
- Abgrenzung der Prozesse
  - `kill(2)`, `ptrace(2)`
  - /etc/init.d/sshd restart
- Abgrenzung des Filesystems
  - Keine Kollisionen bei Verwendung von Standardpfaden
  - Wahren von Geheimnissen

# Anforderungen



- Verhalten wie „normaler“ Server
- Abgrenzung der Prozesse
  - `kill(2)`, `ptrace(2)`
  - /etc/init.d/sshd restart
- Abgrenzung des Filesystems
  - Keine Kollisionen bei Verwendung von Standardpfaden
  - Wahren von Geheimnissen
- Keine Hintertüren
  - direkter Hardwarezugriff (/dev/hda)
  - direkter Kernelzugriff (/dev/kmem)

# Anforderungen



- Verhalten wie „normaler“ Server
- Abgrenzung der Prozesse
  - `kill(2)`, `ptrace(2)`
  - /etc/init.d/sshd restart
- Abgrenzung des Filesystems
  - Keine Kollisionen bei Verwendung von Standardpfaden
  - Wahren von Geheimnissen
- Keine Hintertüren
  - direkter Hardwarezugriff (/dev/hda)
  - direkter Kernelzugriff (/dev/kmem)

$\Rightarrow$ Keine Beeinflussung der Funktion anderer Server oder des Hosts

# Wünsche



- Effizienz
  - Performance (CPU, I/O)
  - Speicher (RAM, Platte)

# Wünsche



- Effizienz
  - Performance (CPU, I/O)
  - Speicher (RAM, Platte)
- leichte Managebarkeit
  - Erstellung
  - Betrieb

  $\Rightarrow$ Nutzung bekannter Mittel

# Wünsche



- Effizienz
  - Performance (CPU, I/O)
  - Speicher (RAM, Platte)
- leichte Managebarkeit
  - Erstellung
  - Betrieb

  $\Rightarrow$ Nutzung bekannter Mittel
- Migration auf andere physikalische Rechner

# Lösungsmöglichkeiten



- vmware/bochs/qemu
  - wie „normaler“ Rechner handhabbar
  - aber: sehr hoher Resourcenverbrauch; meist nur für i386

# Lösungsmöglichkeiten



- vmware/bochs/qemu
  - wie „normaler“ Rechner handhabbar
  - aber: sehr hoher Resourcenverbrauch; meist nur für i386
- UML
  - mittlerer bis hoher Resourcenverbrauch

# Lösungsmöglichkeiten



- vmware/bochs/qemu
  - wie „normaler“ Rechner handhabbar
  - aber: sehr hoher Resourcenverbrauch; meist nur für i386
- UML
  - mittlerer bis hoher Resourcenverbrauch
- SELinux
  - Anforderungen erfüllbar
  - keine vollständige Virtualisierung (Hostname, IP)
  - ???

# Lösungsmöglichkeiten



- vmware/bochs/qemu
  - wie „normaler“ Rechner handhabbar
  - aber: sehr hoher Resourcenverbrauch; meist nur für i386
- UML
  - mittlerer bis hoher Resourcenverbrauch
- SELinux
  - Anforderungen erfüllbar
  - keine vollständige Virtualisierung (Hostname, IP)
  - ???
- Linux vserver, BSD Jails, SUN Zones, FreeVPS
  - Gruppierung von Prozessen
  - Nutzung gemeinsamer Hardware und Kernel
  - Neue und bereits existierende Zugriffsbeschränkungen
  - nahezu kein Overhead



# vserver

# Quickstart (1)



1. Download und Entpacken der Kernel-Quellen:

```
$ wget http://ftp.kernel.org/pub/linux/kernel/v2.x/linux-2.x.y.tar.bz2
$ tar xjf linux-2.x.y.tar.bz2
$ cd linux-2.x.y
```

2. Download des entsprechenden vserver-Patches[a] von http://www.13thfloor.at/vserver/project/ und Anwendung dieses

```
$ bzcat patch-2.x.y-vs1.z.diff.bz2 | patch -p1
```

3. Konfiguration, Bau und Installation des Kernels

```
$ make config
$ make dep && make all modules && make install modules_install
```

[a] 1.2x – stable, 1.3x und 1.9x – experimentell

# Quickstart (2)



4. Download der Userspace-Tools[a] (util-vserver) von http://www.nongnu.org/util-vserver
5. Konfiguration, Bau und Installation dieser mit

```
$ rpmbuild -ta util-vserver-0.x.y.tar.bz2
# rpm -Uvh ...
```

   oder

```
$ ./configure <options>* && make
# make install
```

   Achtung: evtl. Anpassung der Pfade erforderlich
6. Reboot

http://linux-vserver.org

[a] $0.x.y \longrightarrow$ stable wenn kein y, pre bei y$<$90, rc bei 90$\leq$y$<$190 und alpha bei 190$\leq$y

# Eigenschaften (Kernel) (1)



- Entwickler: Herbert Pötzl
- ein Multicall-Syscall für gesamte Funktionalität
- Attribute für Prozesse:
  - numerische Kontext-ID ($xid$)
    $\longrightarrow$ Prozesse mit $xid_1$ für $xid_2$ nicht sichtbar
  - IPs bzw. Netzwerkkontext ($nid$)[2.6]
- Attribute für Prozess-Kontexte:
  - Hostname bzw. kompletter UTS Eintrag[2.6]
  - system- & kontextspezifische[2.6] Capabilities
  - Flags
  - Namespace[2.6]
  - Scheduling Parameter[2.6]

# Eigenschaften (Kernel) (2)



- Sicherheit größtenteils basierend auf Linux-Capabilities (/usr/include/linux/capability.h), z.B.
  - kein Erzeugen neuer Devices ohne CAP_MKNOD
  - keine Interfacekonfiguration ohne CAP_NET_ADMIN
  - kein Mounten ohne CAP_SYS_ADMIN
  - ...
- Verstecken von Filesystemeinträgen
  einige Einträge in /proc ohne Capability-Schutz, z.B. /proc/sysrq-trigger oder /proc/scsi/scsi
  $\Rightarrow$ Verstecken ausserhalb Hostkontext
- Kontext-Quotas
- ausbruchsichere chroot(2) Umgebung

# Exkurs: chroot Attacken (1)

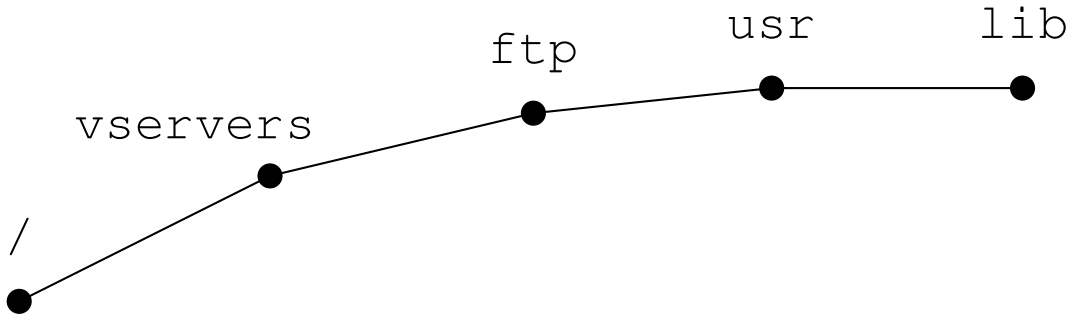


- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen

# Exkurs: chroot Attacken (1)



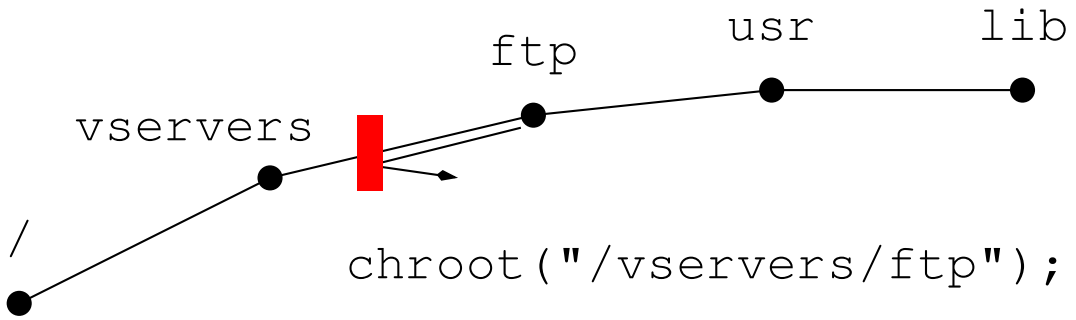
- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:



# Exkurs: chroot Attacken (1)



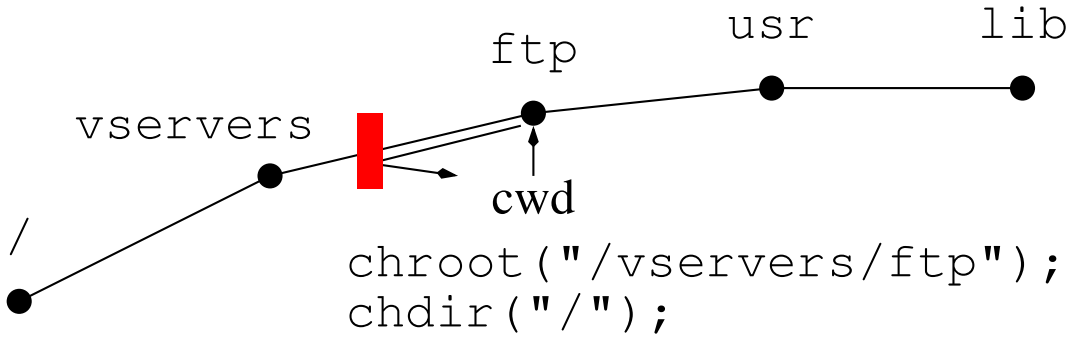
- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:



# Exkurs: chroot Attacken (1)



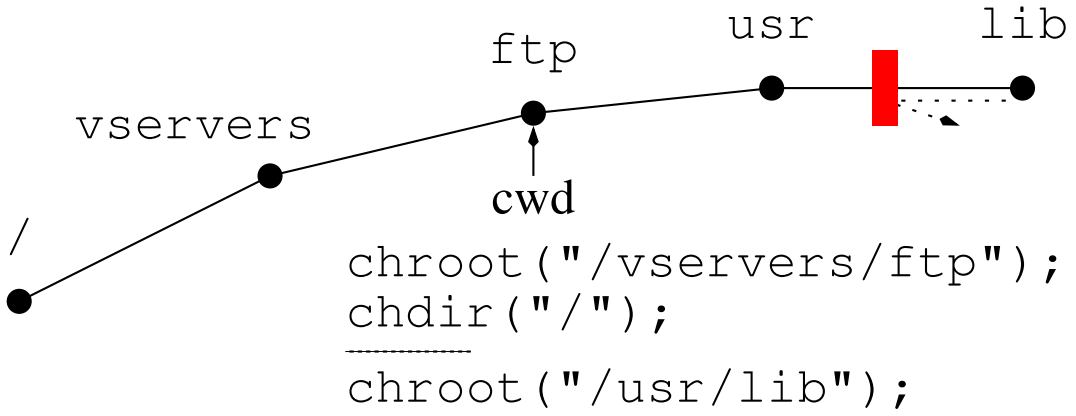
- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:



# Exkurs: chroot Attacken (1)



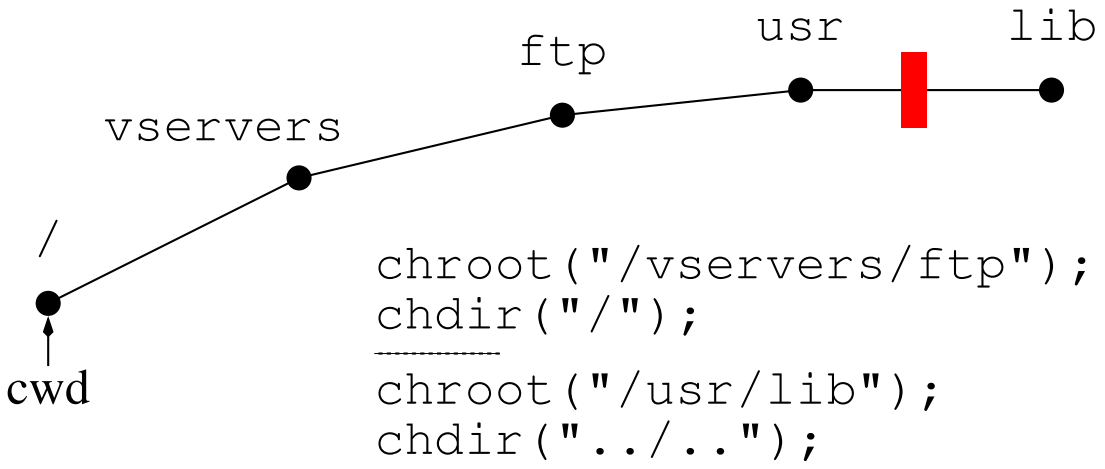
- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:



# Exkurs: chroot Attacken (1)



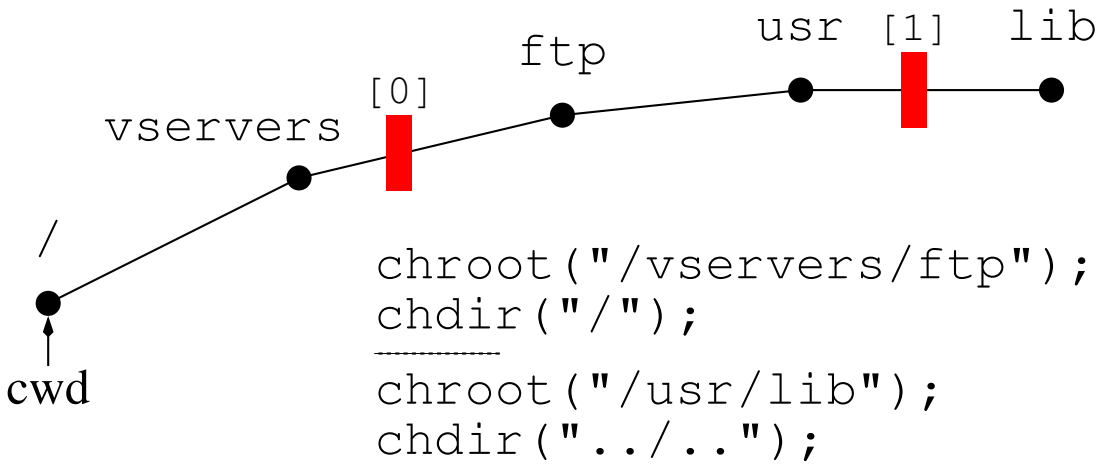
- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:



# Exkurs: chroot Attacken (1)



- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:

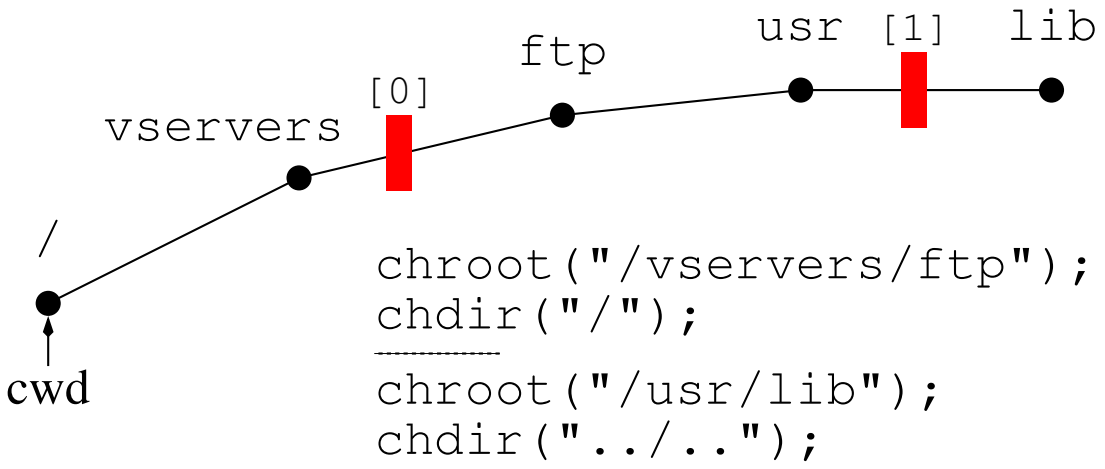



- Offensichtliche Lösungen:
  - Merken vorheriger chroot()'s
  - Verbot von cwd und anderer FDs ausserhalb des chroot's

# Exkurs: chroot Attacken (1)



- chroot(char const *path) $\Rightarrow$ Vermerk der path-Inode als Prozessattribut; ".." tote Zone für Filesystemoperationen
- Beispiel:




- Offensichtliche Lösungen:
  - Merken vorheriger chroot()'s
  - Verbot von cwd und anderer FDs ausserhalb des chroot's
- Wirklich???

# Exkurs: chroot Attacken (2)



- Nein!

# Exkurs: chroot Attacken (2)

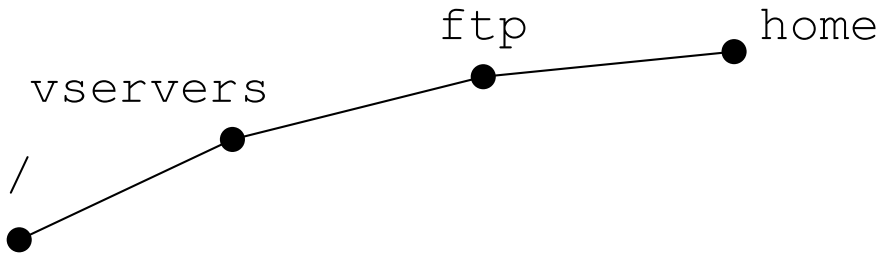


- Nein! FDs immer noch austauschbar (sogar als non-root)

# Exkurs: chroot Attacken (2)



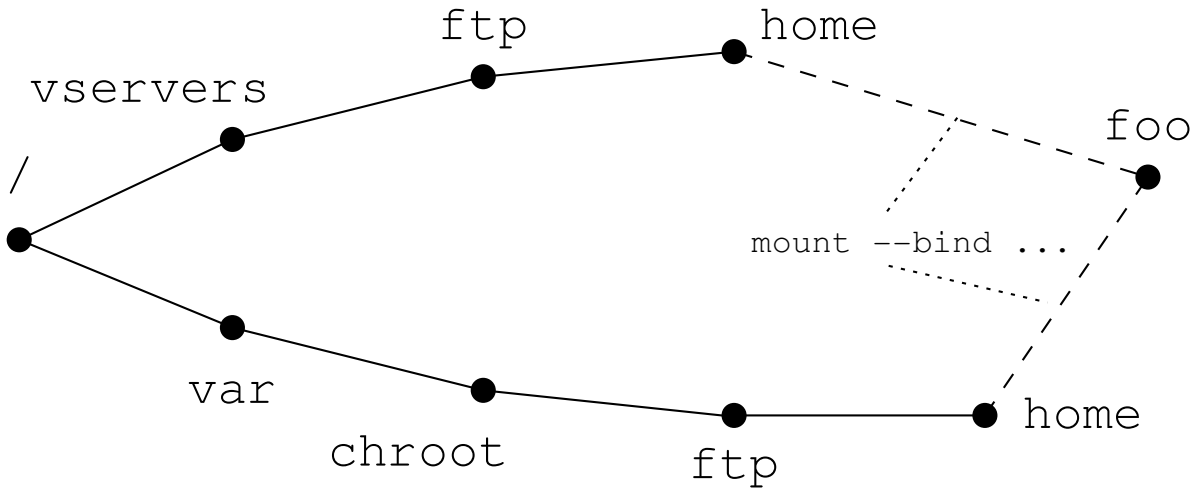
- Nein! FDs immer noch austauschbar (sogar als non-root)



# Exkurs: chroot Attacken (2)



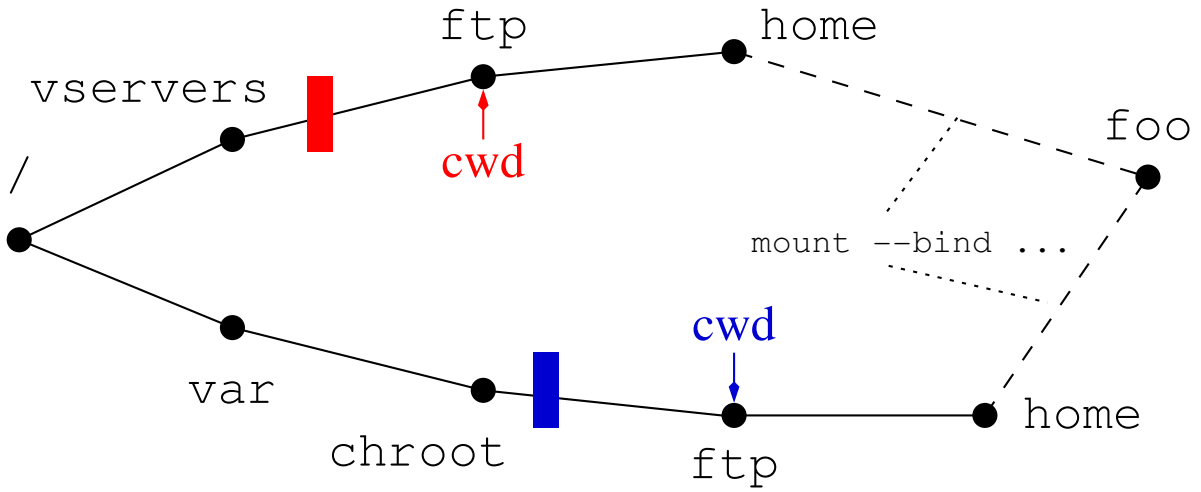
- Nein! FDs immer noch austauschbar (sogar als non-root)



# Exkurs: chroot Attacken (2)



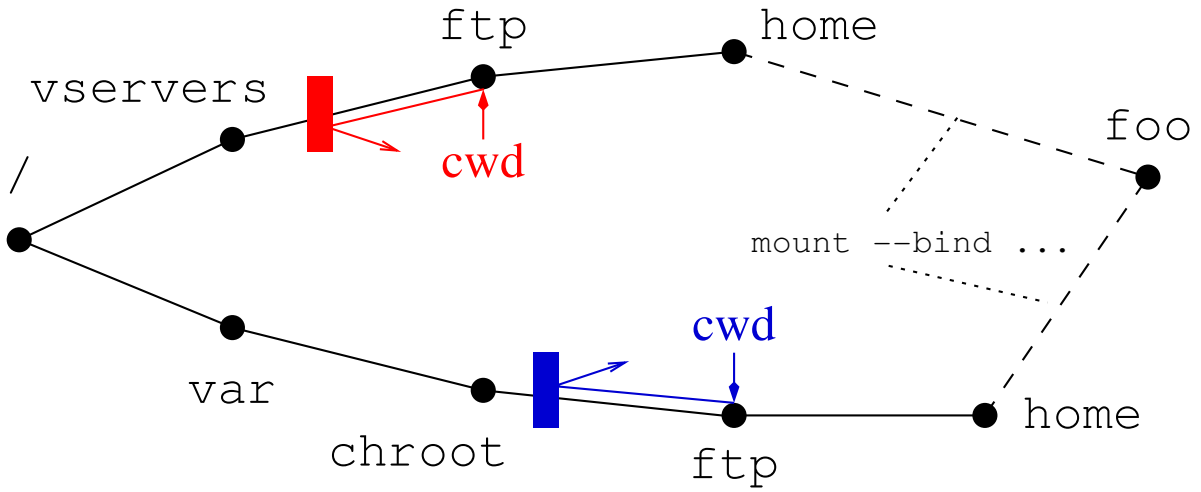
- Nein! FDs immer noch austauschbar (sogar als non-root)



# Exkurs: chroot Attacken (2)



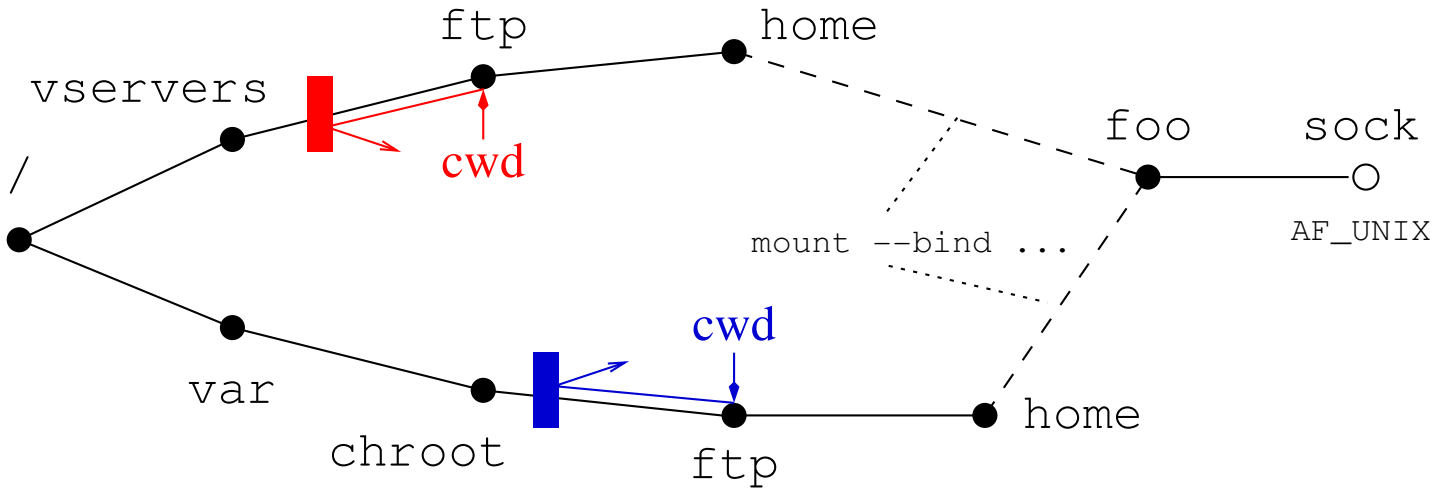
- Nein! FDs immer noch austauschbar (sogar als non-root)



# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)



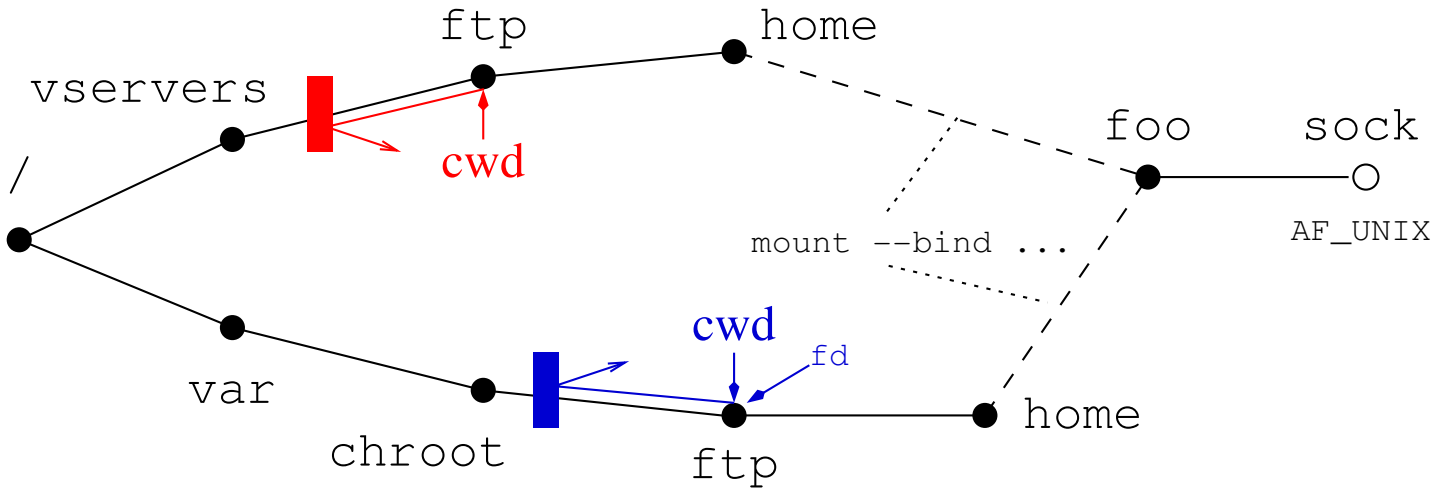

```
tmp=socket(AF_UNIX,...);
bind(tmp,"/home/foo/sock");
listen(tmp)
s=accept(tmp);
```

```
tmp=socket(AF_UNIX,...)
connect(s,"/home/foo/sock");
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)



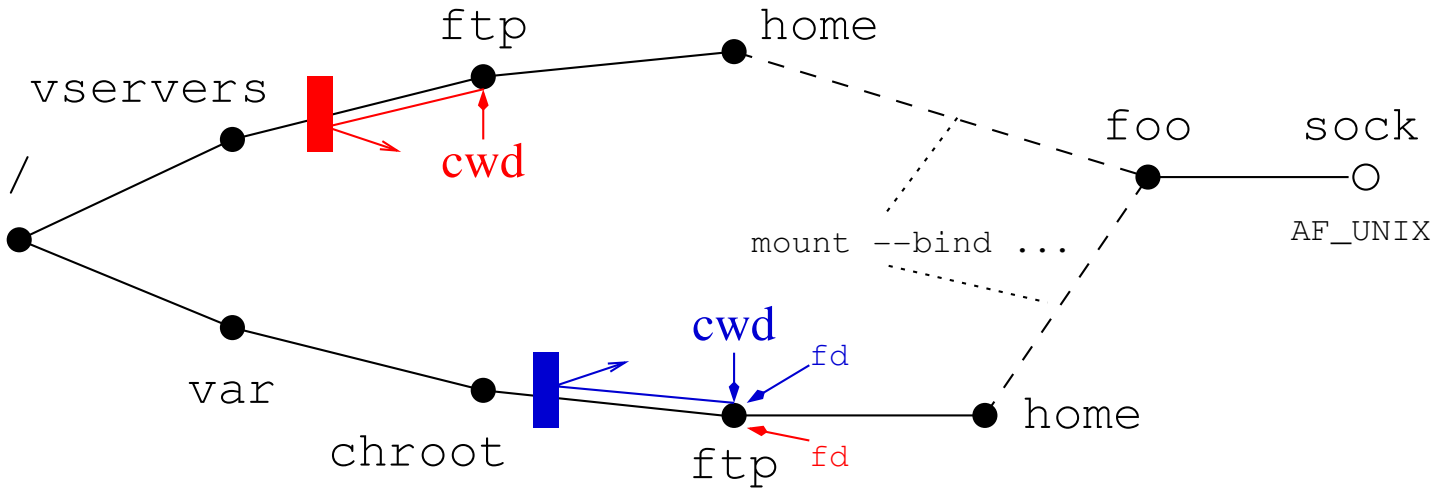

```
tmp=socket(AF_UNIX,...);
bind(tmp,"/home/foo/sock");
listen(tmp)
s=accept(tmp);
```

```
tmp=socket(AF_UNIX,...)
connect(s,"/home/foo/sock");
fd=open(".",O_RDONLY);
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)

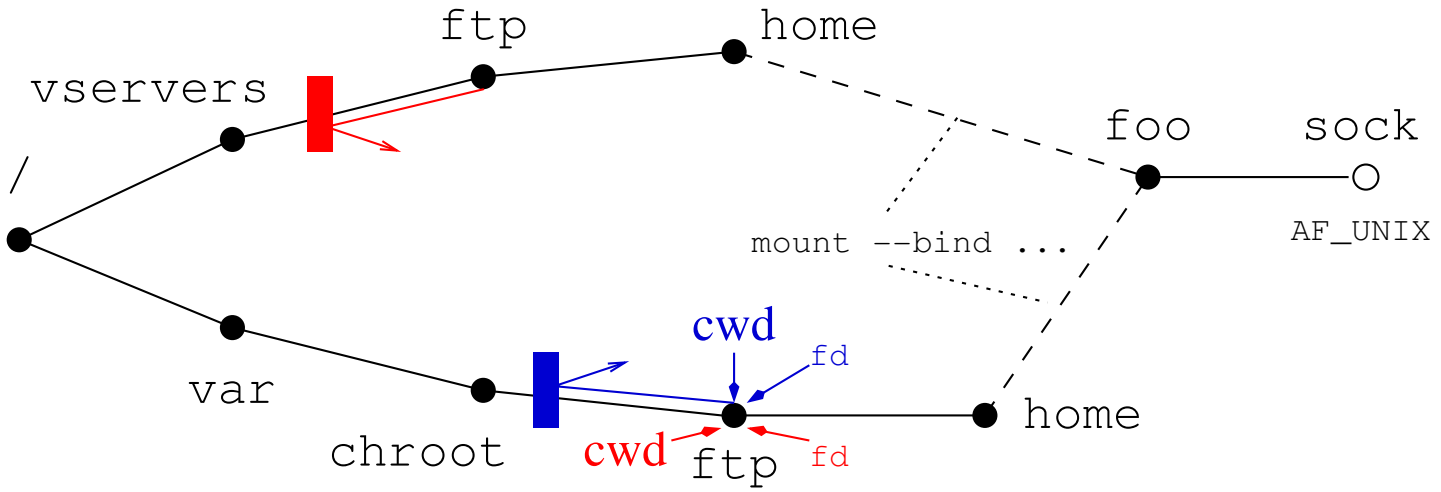



```
tmp=socket(AF_UNIX,...);          tmp=socket(AF_UNIX,...)
bind(tmp,"/home/foo/sock");       connect(s,"/home/foo/sock");
listen(tmp)                       fd=open(".",O_RDONLY);
s=accept(tmp);
recvmsg(s,{&fd,SCM_RIGHTS});      sendmsg(s,{fd,SCM_RIGHTS});
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)

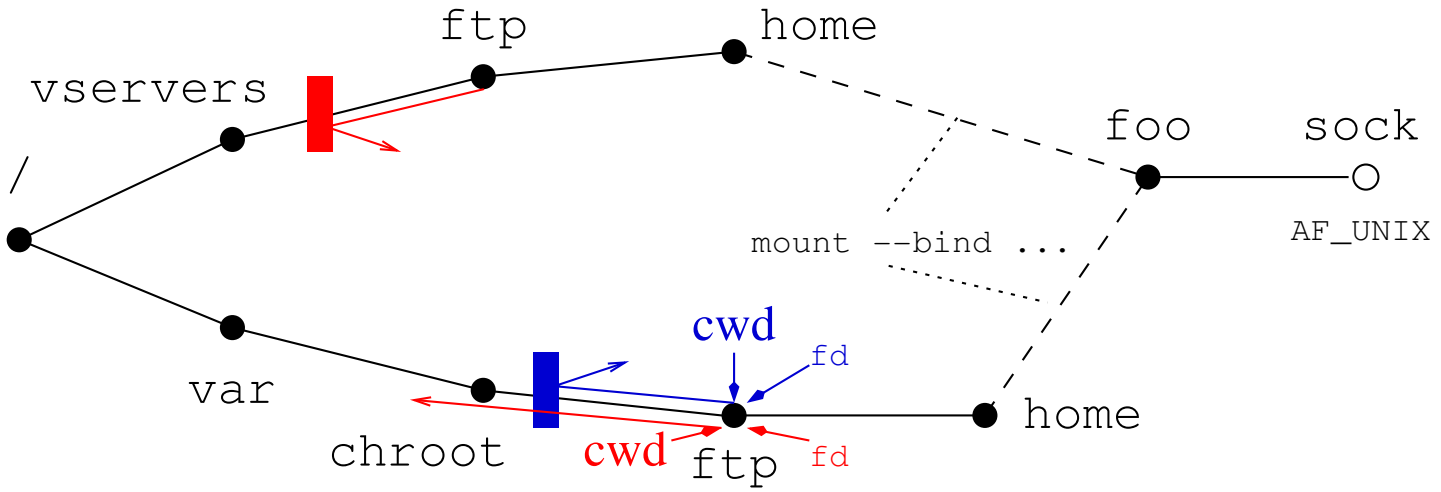



```
tmp=socket(AF_UNIX,...);          tmp=socket(AF_UNIX,...)
bind(tmp,"/home/foo/sock");       connect(s,"/home/foo/sock");
listen(tmp)                       fd=open(".",O_RDONLY);
s=accept(tmp);
recvmsg(s,{&fd,SCM_RIGHTS});      sendmsg(s,{fd,SCM_RIGHTS});
fchdir(fd);
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)

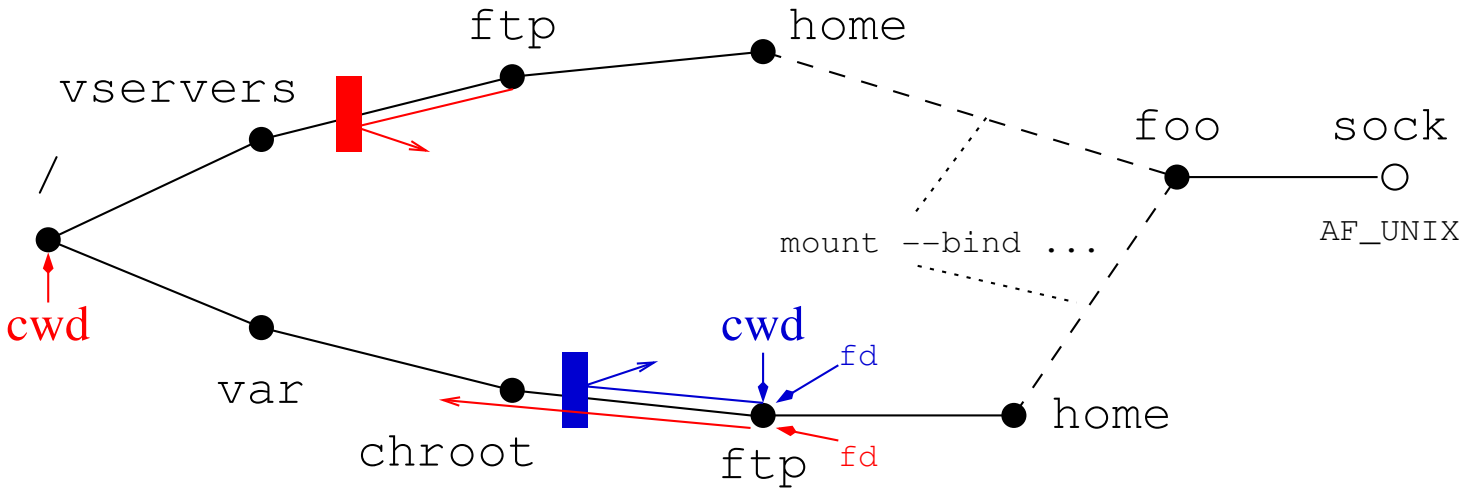



```
tmp=socket(AF_UNIX,...);          tmp=socket(AF_UNIX,...)
bind(tmp,"/home/foo/sock");       connect(s,"/home/foo/sock");
listen(tmp)                       fd=open(".",O_RDONLY);
s=accept(tmp);
recvmsg(s,{&fd,SCM_RIGHTS});      sendmsg(s,{fd,SCM_RIGHTS});
fchdir(fd);
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)

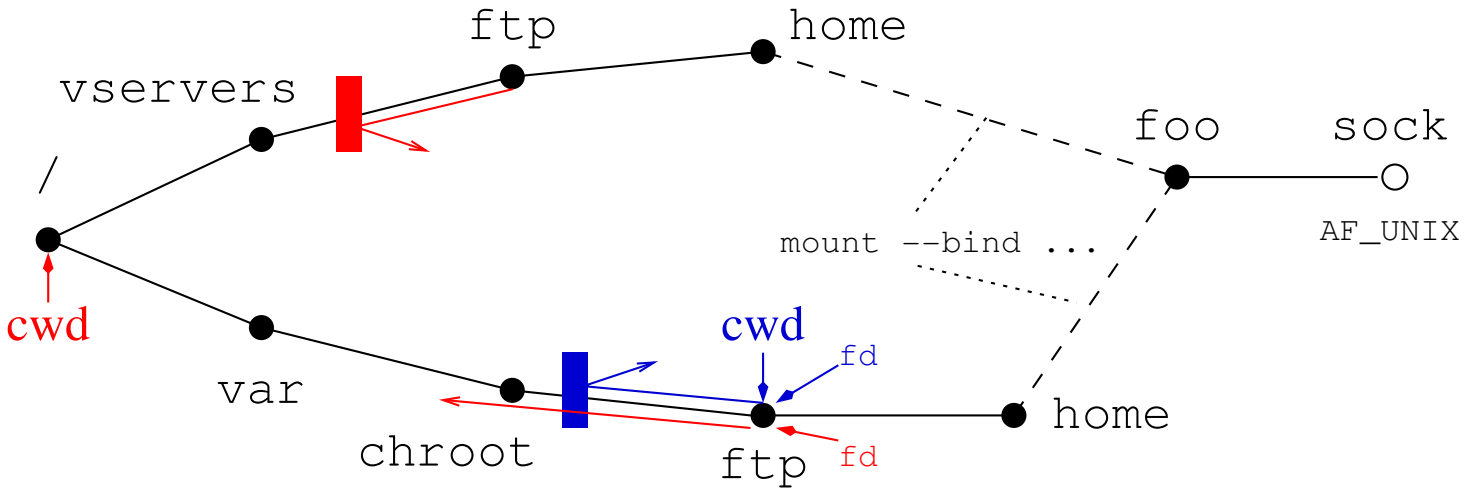



```
tmp=socket(AF_UNIX,...);          tmp=socket(AF_UNIX,...)
bind(tmp,"/home/foo/sock");       connect(s,"/home/foo/sock");
listen(tmp)                       fd=open(".",O_RDONLY);
s=accept(tmp);
recvmsg(s,{&fd,SCM_RIGHTS});      sendmsg(s,{fd,SCM_RIGHTS});
fchdir(fd);
chdir("../../..");
```

# Exkurs: chroot Attacken (2)



- Nein! FDs immer noch austauschbar (sogar als non-root)




```
tmp=socket(AF_UNIX,...);          tmp=socket(AF_UNIX,...)
bind(tmp,"/home/foo/sock");       connect(s,"/home/foo/sock");
listen(tmp)                       fd=open(".",O_RDONLY);
s=accept(tmp);
recvmsg(s,{&fd,SCM_RIGHTS});      sendmsg(s,{fd,SCM_RIGHTS});
fchdir(fd);
chdir("../../..");
```

- Achtung: auch ausserhalb von chroot-Attacken relevant: untrusted roter root in trusted blauen Filesystem

# Exkurs: chroot Attacken (3)



Funktionierende Lösungen:

- Statische Barrieren im Filesystem
  - chmod 000 <*dir*> & `t`-Attribut, bzw.
  - spezielles Barriere Flag in Extended Attributes$^{(2.6)}$

  $\Rightarrow$ blaue chroot Barriere auch für roten Prozess gültig
- Namespaces:
  1. clone(. . . , CLONE_NS, . . . )
  2. mount –rbind /vservers/ftp /

  $\Rightarrow$ kein existierendes „. .“ für chdir(“..”)

  $\rightsquigarrow$ relativ unerprobt; Konflikte mit Automountern
- SELinux
  - laut Russel Coker, sichere chroot’s möglich
  - momentan nicht implementiert

# Userspace



- vserver == chroot-Umgebung + Konfigurationsdaten

# Userspace



- vserver == chroot-Umgebung + Konfigurationsdaten
- Zwei Toolsets: „vserver“ und „util-vserver“
- Low-level Syscallwrappers
- Verwaltung der vserver
  - Erstellung
  - Inbetriebnahme/Stoppen
  - Optimierung

# Userspace



- vserver == chroot-Umgebung + Konfigurationsdaten
- Zwei Toolsets: „vserver“ und „util-vserver“
- Low-level Syscallwrappers
- Verwaltung der vserver
  - Erstellung
  - Inbetriebnahme/Stoppen
  - Optimierung
- chroot-Umgebung gewöhnlich unter /vservers/*<id>*
- Konfiguration unter /etc/vservers/

# Userspace



- vserver == chroot-Umgebung + Konfigurationsdaten
- Zwei Toolsets: „vserver“ und „util-vserver“
- Low-level Syscallwrappers
- Verwaltung der vserver
  - Erstellung
  - Inbetriebnahme/Stoppen
  - Optimierung
- chroot-Umgebung gewöhnlich unter /vservers/*<id>*
- Konfiguration unter /etc/vservers/
- Starten eines Vservers mit „vserver *<id>* start“; Stoppen mit „vserver *<id>* stop“

# util-vserver, stable (1)



- „vserver“ und stable-Branch von „util-vserver“:
  - nahezu identische Funktionalität
  - Fork von „util-vserver“ bei „vserver 0.23“
- weit verbreitet
- sehr gute Dokumentation

# util-vserver, stable (1)



- „vserver“ und stable-Branch von „util-vserver“:
  - nahezu identische Funktionalität
  - Fork von „util-vserver“ bei „vserver 0.23“
- weit verbreitet
- sehr gute Dokumentation
- viele offene Wünsche
- nicht einsetzbar in feindlichen Umgebungen

# util-vserver, stable (1)



- „vserver“ und stable-Branch von „util-vserver“:
  - nahezu identische Funktionalität
  - Fork von „util-vserver“ bei „vserver 0.23“
- weit verbreitet
- sehr gute Dokumentation
- viele offene Wünsche
- nicht einsetzbar in feindlichen Umgebungen:
  rm −f var/lock/subsys/∗

# util-vserver, stable (1)



- „vserver“ und stable-Branch von „util-vserver“:
  - nahezu identische Funktionalität
  - Fork von „util-vserver“ bei „vserver 0.23“
- weit verbreitet
- sehr gute Dokumentation
- viele offene Wünsche
- nicht einsetzbar in feindlichen Umgebungen:
  rm −f var/lock/subsys/∗
  /vservers/foo/var/lock/subsys → /etc

⇒ komplettes Redesign erforderlich

- keine Weiterentwicklung
- keine Unterstützung neuer Features

# util-vserver, stable (2)



## Konfiguration in /etc/vservers/<*id*>.conf

```
IPROOT="192.168.5.32 192.168.5.64"
IPROOTDEV=eth0
S_HOSTNAME=ftp.nowhe.re
ONBOOT=yes
S_DOMAINNAME=
S_NICE=5
S_FLAGS="lock nproc fakeinit"
ULIMIT="-HS -u 200"
S_CAPS=""
```

- bash-Skriptlet; eingebunden mit source
- evtl. weiteres Skript /etc/vservers/<*id*>.sh für Aufgaben nach/vor dem Starten/Stoppen

# util-vserver, alpha



- Designziele:
  - leicht erweiterbar
  - keine Races; resistent gegen Symlinkangriffe
  - Eingebaute Lösungen für Standardaufgaben
  - Unterstützung neuer Kernelfeatures
- Beibehaltung der Basiskommandos von stable, aber viele neue Befehle und Reimplementierung alter
- Neues Konfigurationsschema

http://www.linux-vserver.org/index.php?page=alpha+util-vserver

# util-vserver, alpha



- Designziele:
  - leicht erweiterbar
  - keine Races; resistent gegen Symlinkangriffe
  - Eingebaute Lösungen für Standardaufgaben
  - Unterstützung neuer Kernelfeatures
- Beibehaltung der Basiskommandos von stable, aber viele neue Befehle und Reimplementierung alter
- Neues Konfigurationsschema
- kaum Dokumentation

http://www.linux-vserver.org/index.php?page=alpha+util-vserver

# Sicherheit



- Verhinderung von Symlinkangriffen durch *sicheres* Betreten der Verzeichnisse; meist implementiert durch

```
chroot(vserver_rootdir);
chdir(destination_directory);
action();
```

- Durchführen von Operationen nur in "." (exec-cd Tool)

```
// Usage: exec-cd <dir> <cmd> <args>*
old_fd = open("/", O_RDONLY);
chroot(".");
chdir(argv[1]);
new_fd = open(".", O_RDONLY);
fchdir(old_fd);
chroot(".");
fchdir(new_fd);
execv(argv[2], argv+2);
```

- kein Vertrauen in Daten innerhalb des vservers; z.B. Paketmanagement-Metadaten (rpmdb, apt-Listen) komplett ausserhalb des vservers

# Konfiguration



- Konfiguration in /etc/vservers/<*id*>/ **Verzeichnis**

```
/etc/vservers/ftp
|-- capabilities
|-- context
|-- flags
|-- fstab
|-- interfaces
|   |-- 00
|   |   |-- ip
|   |   `-- name
|   |-- bcast
|   |-- dev
|   `-- mask
|-- run -> /var/run/vservers/ftp
|-- run.rev -> ../.defaults/run.rev
`-- vdir -> /etc/vservers/.defaults/vdirbase/ftp
```

- Dateien & Symlinks; meist Ein-Eintrag-pro-Zeile/Datei
- Identifikation eines vservers durch Pfad des Konfigurationsverzeichnisses; chroot-Pfad frei wählbar
- nur formelle Dokumentation

# vcontext (1)



- früher: chcontext, aber keine Unterstützung neuer Technologien
- Erzeugung (`--create`) und Betreten (`--migrate`) von Prozess-Kontexten
- Beim Betreten auf Sicherheit achten! (ptrace(2))
- Zwischen Erzeugung und Betreten gewöhnlich noch andere Operationen
- Aufruf gewöhnlich als:

```
vcontext --create -- \
  vattribute --set -- \
  vlimit ... -- \
  vsched ... -- \
  vcontext --migrate-self --endsetup -- \
  <command>
```

# vcontext (2)



## Beispiel:

```
# vcontext --create ps axh
New security context is 49153
 5440 pts/1    R      0:00 ps axh

# vcontext --migrate --xid 43 ps axh
 5068 ?        S      0:00 /sbin/syslogd
 5102 ?        S      0:00 /usr/sbin/exim4 -bd -q30m
 5108 ?        S      0:00 /usr/sbin/inetd
 5112 ?        S      0:00 /usr/sbin/atd
 5115 ?        S      0:00 /usr/sbin/cron
 5447 pts/1    R      0:00 ps axh
```

# vattribute



- Setzen/Löschen von Attributen und Capabilities
- Darstellungsmöglichkeiten dieser Werte:
  - Zeichenkette
  - Zahl: Interpretation als Bitmuster
  - Prefix ‘~’ oder ‘!’: Löschen des entsprechenden Bitmusters
  - Prefix ‘^’: Interpretation als Bitnummer

## Beispiel:

```
# vcontext --create vattribute --set --flag hidemount cat /proc/mounts
# vcontext --create -- \
    vattribute --set --secure -- \
    vcontext --endsetup --migrate-self -- \
    mknod /tmp/test c 1 2
New security context is 49183
mknod: ‘/tmp/test’: Operation not permitted
```

# chbind



- Binden von IPs an Prozesse
- bald: Ablösung durch vnet

## Beispiel:

```
# chbind --ip 10.1.0.1 cat /proc/self/status | grep ipv4root
ipv4root is now 10.1.0.1
ipv4root: 0100010a/00ffffff
ipv4root_bcast: ffffffff
ipv4root_refcnt: 2
```

# Verschiedenes (1)



## vkill

- Atomares Senden von Signalen an Prozess-Kontext

## vnamespace

- Erzeugen und Betreten von Namespaces

## vlimit

- Setzen & Anzeigen von Beschränkungen

# Verschiedenes (2)



## vuname

- Anzeigen und Ändern von uts-Einträgen

## vserver-info

- Abfrage einzelner Attribute von Kontexten und vservern
- wichtig für Bugreports:
  vserver −info − SYSINFO



# Management

# vserver Erzeugung



- keine „normale“ Systeminstallation möglich
- BSD Jails:

```
# make -C /usr/src DESTDIR=/vservers/foo install
```

# vserver Erzeugung



- keine „normale“ Systeminstallation möglich
- BSD Jails:

```
# make -C /usr/src DESTDIR=/vservers/foo install
```

- Fedora Core:

```
# make -C /usr/src DESTDIR=/vservers/foo install
make: Entering directory ‘/usr/src’
make: *** No rule to make target ‘install’.  Stop.
make: Leaving directory ‘/usr/src’
```

- $x$ Distributionen mit $x + n$ Installationsmöglichkeiten

# vserver Erzeugung



- keine „normale“ Systeminstallation möglich
- BSD Jails:

```
# make -C /usr/src DESTDIR=/vservers/foo install
```

- Fedora Core:

```
# make -C /usr/src DESTDIR=/vservers/foo install
make: Entering directory '/usr/src'
make: *** No rule to make target 'install'.  Stop.
make: Leaving directory '/usr/src'
```

- $x$ Distributionen mit $x + n$ Installationsmöglichkeiten

$\Rightarrow$ Implementierung *einiger* in util-vserver:

- „apt-rpm“ für Fedora/RH vserver
- „debootstrap“ für Debian vserver
- „skeleton“ für Basis Verzeichnisstruktur + Konfiguration

# vserver . . . build



- Dokumentation durch „vserver - build --help“
- Beispiel:
  - ```
    # vserver test0 build -m apt-rpm --hostname test0.nowhe.re \
      --interface 10.0.1.0 --netdev eth0 --netprefix 23 \
      --context 42 -- -d fc1
    ```
  - ```
    # vserver test1 build -m debootstrap --hostname test1.nowhe.re \
      --interface 10.0.1.1 --netdev eth0 --netprefix 23 \
      --context 43 -- -d sarge
    ```
  - ```
    # vserver test2 build -m skeleton --hostname test2.nowhe.re \
      --interface 10.0.1.2 --netdev eth0 --netprefix 23 \
      --context 44
    ```
- Konfiguration der Parameter (Mirror, Paketlisten) durch /etc/vservers/.defaults/apps/debootstrap/* und /etc/vservers/.distributions/*
    ```

# vserver . . . start



1. Erzeugung eines Namespaces
2. Erzeugung von Netzwerkinterfaces
3. Mounten von Verzeichnissen
4. Erzeugung von Prozess- und Netzwerkkontext
5. Aktivierung von Beschränkungen (Capabilities, Limits)
6. Aufruf des init-Prozesses:
   - Shortcut über „/etc/rc.d/rc 3“, oder
   - reguläres /sbin/init – oft Menge unerwünschter Aktionen

   $\Rightarrow$ `fakeinit` Mechanismus nötig (getpid()==1)

Achtung: mindestens ein laufender Prozess benötigt

## Beispiel:

```
# vserver test0 start
# vserver --debug test1 start
```

# vserver . . . stop



- entweder
  - Senden von `SIGINT` an init-Prozess, oder
  - Ausführen von „/etc/rc.d/rc 6“
- explizites „vkill –xid *<xid>* -s 9“
- Rückgängigmachen von „vserver . . . start“
- bei Verwendung von Namespaces kein explizites Unmounten nötig

## Beispiel:

```
# vserver test0 stop
# vserver --debug test1 stop
```

# vserver . . . enter|exec



- Ausführen von Kommandos innerhalb des vservers
- Ähnliche Aktionen wie bei „vserver . . . start“, aber Betreten statt Erzeugen von Namespaces und Kontexten

⇒ kein Überschreiben eventueller Parameter und Restriktionen

- möglichst nur für Administrationsaufgaben, nicht für regulären Betrieb (z.B. fehlende /dev/pts Einträge)

## Beispiel:

```
# vserver test0 exec ps axh
  640 ?        S      0:00 syslogd -m 0
 1188 pts/1    R      0:00 ps axh
# vserver test1 enter
test1:/#

# uname -a
Linux delenn 2.6.5ensc-0.3 #1 Thu Apr 15 ... 2004 i686 i686 i386 GNU/Linux
# vserver test1 exec uname -a
SCO UnixWare test1.nowhe.re 7.1 #1 Sat Feb 29 ... 2003 s390 GNU/Linux
```

# vps



- Anzeige aller Prozesse auf dem Host
- Ausführung von ps in speziellem Watcher-Kontext (XID 1)

## Beispiel:

```
# vps ax
  PID CONTEXT             TTY       STAT   TIME COMMAND
    1       0 MAIN        ?         S      0:05 /sbin/minit
    2       0 MAIN        ?         SWN    0:00 [ksoftirqd/0]
 ...
 5068      43 test1       ?         S      0:00 /sbin/syslogd
 5102      43 test1       ?         S      0:00 /usr/sbin/exim4 -bd -q30m
 5108      43 test1       ?         S      0:00 /usr/sbin/inetd
 5112      43 test1       ?         S      0:00 /usr/sbin/atd
 5115      43 test1       ?         S      0:00 /usr/sbin/cron
 ...
 5256      42 test0       ?         S      0:00 syslogd -m 0
 ...
 5276       1 ALL_PROC    pts/1     S      0:00 vps ax
 5277       1 ALL_PROC    pts/1     R      0:00 ps ax
```

# vserver-stat



- Überblick über laufende vserver bzw. Prozess-Kontexte

## Beispiel:

```
# vserver-stat
CTX   PROC    VSZ    RSS  userTIME   sysTIME    UPTIME NAME
0       37  43.8M   4.6K   0m37s86   0m22s52  13h00m44 root server
42       1   1.5M   145    0m00s00   0m00s00   2m56s60 test0
43       5  10.5M   965    0m00s10   0m00s00   8m53s31 test1

# vserver-stat
CTX   PROC    VSZ    RSS  userTIME   sysTIME    UPTIME NAME
0       48 143.9M   3.5K  19h09m59   8h40m24  56d45h05 root server
2        3   6.7M   172    3h01m34   1h22m00  28d29h14 vpn
82      17  90.6M   784    4h44m09   2h13m18  28d26h53 cvs
133      5   1.6M    52   31m03s55   5m46s86  23d33h26 paris
146     11  48.9M     2K  37m07s12   8m55s74  28d26h53 mirror
147      7   3.8M   180   10h46m17   2h48m54  28d21h46 mirror-master
153      9  74.6M     4K  36m31s24   7m08s33  28d25h46 ldap1
```

# vrpm (1)



- externes oder internes Halten der Datenbank
- Vorteil intern: rpm Funktionalität innerhalb des vservers
- Vorteil extern: leichtes Bootstrappen („vserver . . . build“)
- Wechsel zwischen beiden Methoden via
  vserver ... pkgmgmt externalize|internalize

**Syntax:**

vrpm <vserver>+ −− <rpm−options>+

## Internes vrpm:

- Realisiert über „vserver . . . exec rpm“

# vrpm (2)



## Externes vrpm:

- `LD_PRELOAD` Wrapper für execv(3), getpwnam(3) & Co.
  ⇒ Ausführung von %scriplets im Kontext des vservers
  ⇒ NSS Lookups beim Entpacken der Pakete
- kompliziertes Mounten der Datenbank in vserver, so dass Zugriff durch vserver-Prozesse oder %scriptlets unmöglich
- Daten unter /etc/vservers/<*id*>/apps/pkgmgmgt/. . . bzw. /vservers/.pkg/<*id*>/rpm

## Beispiel:

```
# vrpm test0 -- -q glibc fedora-release rpm
glibc-2.3.2-101.4
fedora-release-1-3
package rpm is not installed
# vrpm test0 -- -Uvh /tmp/tetex-2.0.2-13.i386.rpm
```

# vapt-get



- für RPM basierende vserver: externes und internes Management möglich
- sonst realisiert über „vserver . . . exec apt-get“

## Syntax:

vapt−get <vserver>+ −− <apt−get−options>+

## Beispiel:

```
# vapt-get test0 -- install bzip2-libs
...
Preparing...                ########################################### [100%]
   1:bzip2-libs             ########################################### [100%]
Done.

# vapt-get test1 -- install libbz2-1.0
...
Unpacking libbz2-1.0 (from .../libbz2-1.0_1.0.2-1_i386.deb) ...
Setting up libbz2-1.0 (1.0.2-1) ...
```

# setattr, showattr (1)



- meist: Menge von vservern mit gleicher Distribution
  $\Rightarrow$ Installation und Ausführung identischer Pakete, Binaries, Daten
- Idee: Kopieren via Hardlinks („ln A B“)
  $\Rightarrow$ Einsparung von Plattenplatz
  $\Rightarrow$ Einsparung von Speicher (Mapping von Programmen und Libraries)

# setattr, showattr (1)



- meist: Menge von vservern mit gleicher Distribution
  - $\Rightarrow$ Installation und Ausführung identischer Pakete, Binaries, Daten
- Idee: Kopieren via Hardlinks („ln A B“)
  - $\Rightarrow$ Einsparung von Plattenplatz
  - $\Rightarrow$ Einsparung von Speicher (Mapping von Programmen und Libraries)
  - $\rightsquigarrow$ Manipulationen möglich, da Änderungen auf jedem vserver sichtbar:

    **echo** mycode >/usr/sbin/httpd
    - Kein COW oder unionfs unter Linux

# setattr, showattr (2)



- Lösung: spezielles immutable-Flag; z.B. „chattr +i . . . “
  - nicht setzbar ausserhalb von Hostkontext
  - ⇝ Paketmanagement (Updates) nicht mehr möglich

# setattr, showattr (2)



- Lösung: spezielles immutable-Flag; z.B. „chattr +i . . . “
  - nicht setzbar ausserhalb von Hostkontext

  $\rightsquigarrow$ Paketmanagement (Updates) nicht mehr möglich
- weiteres Flag, so dass:
  - Verhinderung von Modifikationen
  - Erlauben von Löschen
- low-level Funktionalität in setattr und showattr Tools
  - Änderung der Modifizierbarkeit mit „–iunlink“
  - Änderung der Sichtbarkeit
  - Setzen des chroot-Barriere Flags

  $\Rightarrow$ „setattr --help“

# setattr, showattr (3)



## Beispiel:

```
# touch /vservers/test0/{a,b,c}
# ln /vservers/test0/{a,b,c} /vservers/test1/
# setattr --iunlink /vservers/test0/a
# chattr +i /vservers/test0/b
# showattr /vservers/test0/{a,b,c}
---bUI- /vservers/test0/a
---buI- /vservers/test0/b
---bui- /vservers/test0/c

# vserver test0 enter
[root@test0]# echo a>a
bash: a: Permission denied
[root@test0]# echo a>b
bash: a: Permission denied
[root@test0]# echo a>c
[root@test0]#

[root@test0]# rm -f a b c
rm: cannot remove ‘b’: Operation not permitted

# vserver test1 enter
test1:/# cat /c
a
```

# vunify



- Fortsetzung des setattr-Konzepts, aber auf ganze Verzeichnisbäume
- Funktion:
  1. Finden gleicher Dateien
  2. Setzen des `iunlink` Flags
  3. Erzeugung eines Hardlinks
- Verwendung statischer Exclude-Listen, und Informationen des Paketmanagements über Konfigurationsdateien

# vunify



- Fortsetzung des setattr-Konzepts, aber auf ganze Verzeichnisbäume
- Funktion:
  1. Finden gleicher Dateien
  2. Setzen des `iunlink` Flags
  3. Erzeugung eines Hardlinks
- Verwendung statischer Exclude-Listen, und Informationen des Paketmanagements über Konfigurationsdateien
- vollständige Fedora Core 1 Installation nur ca. 30 MB nicht teilbarer Dateien

  $\Rightarrow$ 2.6 GB Plattenplatz für 20 vserver á 2 GB

# Referenzen



- Projekthomepage http://linux-vserver.org
- util-vserver http://www.nongnu.org/util-vserver
- #vserver auf oftc.net

# Schluss