Chroot für alle Dienste unter Linux nutzen

by Mark Nielsen (homepage)

About the author:
Mark arbeitet als freischaffender IT Consultant and verbringt Zeit mit seinem Engagement für Dinge wie GNUJobs.com, Artikel und Freie Software schreiben und er arbeitet als Freiwilliger bei eastmont.net mit.

Abstract:

Über chroot abgesicherte Systemdienste verbessern die Systemsicherheit, da bei erfolgreichem Einbruch in das System der mögliche Schaden begrenzt wird.

Einleitung

Was is chroot? Chroot definiert das Universium für ein Programm neu. Genauergesagt, es definiert die Verzeichniswurzel ("ROOT: /") für Programme und Login−Zugänge neu. Für das Programm bzw. die Eingabeaufforderung, die mit chroot eingeschränkt wird, existieren die Verzeichnisse außerhalb nicht.

Wozu ist das nützlich? Wenn jemand in einen Computer einbricht, sieht er nicht alle Dateien des Systems. Auf Dateien, die man aber nicht kennt, kann man nur schwer Kommandos anwenden und man kann sich weiterhin nicht den Inhalt jeder unsicheren Datei anzeigen lassen. Der einzige Nachteil ist, dass dies die Einbrecher nicht davon abhalten dürfte, den Netzverkehr und andere Dinge weiterhin zu belauschen. Man sollte deshalb noch weitere Sicherheitsmaßnahmen ergreifen, die aber nicht Umfang dieses Artikels sein können:

- Absicherung der Netzwerk−Ports
- alle Dienste sollten unter einem nicht Root Account laufen. Weiterhin sollte jeder Dienst über chroot abgesichert sein.
- Systemlog−Dateien sollten auf einen anderen Computer weitergeleitet werden.
- Analyse der Log−Dateien
- Analyse über die Leute, die nach offenen Ports des Systems suchen
- CPU und Speicher Ressourcen für jeden Dienst begrenzen
- Quotas für jeden Account aktivieren

Der Grund, warum ich einen über chroot in Verbindung mit einem nicht Root Account abgesicherten Dienst als eine Verteidigungslinie betrachte ist der, dass wenn jemand in das System als nicht Superuser einbricht und keine Dateien vorfindet, die ihm den root Account ermöglichen, er nur Schaden in seinem kleinen Bereich anrichten kann. Wenn darüberhinaus die meisten Dateien im Bereich dem Nutzer root gehören,

bestehen noch weniger Angriffsmöglichkeiten. Natürlich ist etwas nicht richtig, wenn überhaupt ein Einbruch möglich ist, aber wenn es schon passiert ist, sollte man die möglichen Folgen versuchen zu mildern.

BITTE DRANDENKEN, dass mein Vorgehen nicht unbedingt 100%ig genau ist. Es ist mein erster Versuch sowas darzustellen, aber wenn der Hauptteil funktioniert, sollte der Feinschliff für jeden möglich sein. Das hier ist eigentlich nur die Roadmap für ein chroot HowTo, an der ich arbeite.

Wie können wir chroot anwenden?

Dazu müssen wir ein "/chroot" Verzeichnis anlegen und alle unsere Dienste darunter im folgenden Format ablegen:

- Syslogd wird für jeden Dienst bereitgestellt
- Apache installieren unter /chroot/httpd
- Ssh installieren unter /chroot/sshd.
- PostgreSQL installieren unter /chroot/postmaster.
- Sendmail kann ebenfalls mit chroot gesichert werden, aber es kann leider nicht unter einem nicht Superuser Account ausgeführt werden.
- ntpd installieren unter /chroot/ntpd
- named installieren unter /chroot/named

Jeder Dienst muss komplett isoliert werden.

Mein Perl Skript um chroot Umgebungen zu erzeugen

Config_Chroot.pl.txt sollte nach dem Herunterladen in Config_Chroot.pl umbenannt werden. Dieses Perl Skript zeigt die installierten Dienste an, man kann in die Konfigurationsdateien schauen, die Dienste konfigurieren, starten und anhalten. Das ist im Allgemeinen die Vorgehensweise.

1. chroot Verzeichnis anlegen:
   mkdir −p /chroot/Config/Backup
2. Download von Config_Chroot.pl.txt als /chroot/Config_Chroot.pl
3. Die $Home Variable im Perl Skript muss angepasst werden, wenn /chroot nicht als Home Verzeichnis verwendet wird.
4. Download meiner Konfigurationsdateien

Ganz wichtig: **Ich habe das nur unter RedHat 7.2 und RedHat 6.2 getestet**.

Deshalb muss man eventuell das Perl Skript an die eigene Distribution anpassen.

Zuerst schrieb ich an einem gigantisch riesigen Artikel zu chroot, aber mit dem Perl Skript konnte ich ihn wesentlich kürzen. Denn nachdem ich viele Dienste mit chroot abgesichert hatte, stellte ich fest, dass viele Ähnlichkeiten bestehen. Der einfachste Weg herauszufinden, welche Dateien angepasst werden müssen, ist in die entsprechende manpage zu schauen und "ldd /usr/bin/file" bei Programmen, die Bibliotheken benutzen, einzugeben. Man kann natürlich auch den Dienst einfach mit chroot in eine neue Umgebung zwingen und schauen, welche Fehlermeldungen z.B. in den Log−Dateien angezeigt werden.

Um einen Dienst abzusichern, sollte man folgendermaßen vorgehen:

```
cd /chroot
./Config_Chroot.pl config  SERVICE
./Config_Chroot.pl install SERVICE
```

```
./Config_Chroot.pl start   SERVICE
```

Chroot für Ntpd

Ntpd is ein Dienst zur Synchronisation des eigenen und anderer Computer mit der realen Zeit. Die chroot Absicherung war trivial.

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
#./Config_Chroot.pl config  ntpd
./Config_Chroot.pl install ntpd
./Config_Chroot.pl start   ntpd
```

Chroot für DNS oder named

Bereits ausführlich beschrieben, deshalb
http://www.linuxdoc.org/HOWTO/Chroot−BIND8−HOWTO.html
oder
http://www.linuxdoc.org/HOWTO/Chroot−BIND−HOWTO.html.

Oder man nutzt mein Skript:

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
#./Config_Chroot.pl config  named
./Config_Chroot.pl install named
./Config_Chroot.pl start   named
```

Chroot für Syslog mit Diensten und einigen Kritikpunkten von mir

Ich wollte syslogd auch absichern. Das Problem ist, dass syslogd per Grundeinstellung /dev/log benutzt, welches natürlich in der chroot Umgebung nicht sichtbar ist. Deshalb ist chroot für syslogd nicht einfach möglich. Hier trotzdem einige Lösungsvorschläge:

- Chroot Umgebung für syslogd mit jedem Dienst. Es ist dann zwar möglich Log−Dateien zu erstellen, aber mir gefällt die Lösung nicht, da syslogd als Superuser laufen muss.
- vielleicht kann man ein anderes System zum Erstellen der Log−Dateien nutzen.
- Man kann natürlich auch Daten einfach in Dateien loggen lassen ohne syslogd zu verwenden. Das hat den Nachteil, dass eine Manipulation der Dateien durch einen Einbrecher möglich ist.
- Den Hauptdämon von syslogd so konfigurieren, dass er an verschiedenen Stellen nach allen Diensten schaut. Dazu muss man die −a Option von syslogd nutzen.

Meine einzige Lösung war, syslogd mit in jede chroot Umgebung aufzunehmen. Lieber wäre mir eine Lösung, die als nicht root unter Verwendung einer chroot Umgebung die Daten logged − z.B. auf einen Netzwerk Port. Das ist vielleicht möglich, doch ich habe meine Anstrengungen erst einmal zurückgestellt und werde später daran weiterbasteln.

Möchte man nicht je einen seperaten syslogd für jeden Dienst, dann sollte man folgende Zeile beim Start von syslogd hinzufügen:

```
syslogd -a /chroot/SERVICE/dev/log
```

Hätte ich z.B. ssh und dns laufen, würde es so aussehen:

```
syslogd -a /chroot/ssh/dev/log -a /chroot/named/dev/log -a /dev/log
```

Letzter Hinweis zu syslogd: ich wünschte, es wäre möglich, es unter einem nicht priviligierten Account laufen zu lassen. Ich habe verschiedene Sachen probiert, aber letztendlich aufgeben müssen. Wenn ich syslogd aber dazu bewegen könnte, wäre ein weiteres Sicherheitsrisiko beseitigt und ich würde mich wohler fühlen. Das Gleiche gilt für ein mögliches loggen per externen Rechner.

Chroot für Apache

Das war extrem einfach zu bewerkstelligen. Nachdem ich die Einstellungen vorgenommen hatte, konnte ich Perl Skripte ausführen. Meine Konfigurationsdatei ist sehr lang, da ich PostgreSQL und Perl in meine chroot Umgebung mit aufgenommen habe. Angemerkt sei, dass die Datenbank über das lokale Loopback Device (127.0.0.1) verbunden werden sollte und man diesen Host in den Perl Skripten für das DBI Modul angeben sollte. Hier ist ein Beispiel, wie man den Zugang unter Nutzung von persistenter Verbindungen mit Apache herstellen kann:

```
$dbh ||= DBI->connect('dbi:Pg:dbname=DATABASE',"","", {PrintError=>0});

if ($dbh ) {$dbh->{PrintError} = 1;}
else
  {$dbh ||= DBI->connect('dbi:Pg:dbname=DATABASE;host=127.0.0.1',"","",
      {PrintError=>1});}
```

Quelle: http://httpd.apache.org/dist/httpd/

Zuerst Apache auf dem Hauptsystem unter /usr/local/apache kompilieren und installieren. Danach das Perl Skript nutzen:

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
# ./Config_Chroot.pl config  httpd
./Config_Chroot.pl install httpd
./Config_Chroot.pl start   httpd
```

Meine httpd.conf Datei habe ich ebenfalls anpassen müssen:

```
ExtendedStatus On

<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
</Location>

<Location /server-info>
    SetHandler server-info
    Order deny,allow
    Deny from all
```

```
    Allow from 127.0.0.1
</Location>
```

Jetzt nur noch mit dem Web−Browser http://127.0.0.1/server−status oder http://127.0.0.1/server−info aufrufen und überprüfen!

Chroot für Ssh

Zuerst sollte man ssh vom eigentlichen Port 22 auf Port 2222 umleiten lassen. Jetzt sollte ssh beim Start an Port 2222 unter einem nicht root Account lauschen. Für den ersten Verbindungsaufbau stellen wir sichere Zugänge zur Verfügung, damit ein Login möglich wird, mehr aber auch nicht. Nach dem Login werden sie an ein 2. ssh Programm weitergeleitet, dass an 127.0.0.1:2322 lauscht und den Zugang zum eigentlichen System zur Verfügung stellt. Das 2. ssh Programm sollte wirklich NUR mit dem Loopback Device verbunden sein! Das sollte man tun. Wir werden das an dieser Stelle aber nicht tun. Wir werden lediglich ssh in einer chroot Umgebung als Beispiel installieren. Für den Leser bleibt als Übung die Einrichtung von ssh, so dass es unter einem nicht priviligierten Account läuft und einen 2. ssh Dämon zu installieren, der am Loopback Device lauscht, um Zugang zum realen System zu gewähren.

Es sei nochmal darauf hingewiesen, dass wir nur ssh absichern und jeder sich selber über die Konsequenzen im Klaren sein muss (ein Zugang zum System ist nach der Änderung nicht mehr möglich). Weiterhin interessant wäre, die Log−Dateien auf einem anderen Rechner speichern zu lassen. Weiterhin sollte man OpenSSH verwenden, auch wenn ich hier die kommerzielle Variante benutzt habe aus Gründen der Einfachheit (eine schlechte Ausrede, ich weiß).

Quelle: http://www.ssh.com/products/ssh/download.cfm

Installation von ssh unter /usr/local/ssh_chroot. Dann das Perl Skript:

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
 # ./Config_Chroot.pl config  sshd
./Config_Chroot.pl install sshd
./Config_Chroot.pl start   sshd
```

Ein wirklicher Vorteil einer abgesicherten chroot Umgebung für ssh ist, wenn man ssh als ftp Ersatz nehmen will und demnach nur einen begrenzten Bereich sichtbar machen möchte. Rsync und SCP ergänzen sich wunderbar, um Leute Dateien hochladen zu lassen. Ich drücke mich eigentlich immer um die Installation eines FTP Servers, in den sich Leute einloggen können. Man kann natürlich auch einen FTP Server mit chroot absichern, aber die Passwörter werden dann immer noch im Klartext übertragen, was ich nicht als sehr vorteilhaft empfinde.

Chroot für PostgreSQL

Das war so einfach wie Perl, lediglich einige Bibliotheken werden mehr gebraucht. Ein weiteres kleines Problem war, PostgreSQL für das Netzwerk, dort aber lediglich das Loopback Device, zu öffnen. Nachdem man PostgreSQL abgesichert hat, können andere Dienste darauf aber auch nicht mehr zugreifen, wie z.B. der Apache Server. Ich habe PostgreSQL mit Perl Unterstützung kompiliert, deshalb musste ich viele Perl spezifische Sachen hinzufügen.

Quelle: ftp://ftp.us.postgresql.org/source/v7.1.3/postgresql−7.1.3.tar.gz

PostgreSQL unter /usr/local/postgres kompilieren und installieren. Dann noch das Perl Skript:

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
# ./Config_Chroot.pl config  postgres
./Config_Chroot.pl install postgres
./Config_Chroot.pl start   postgres
```

Chroot für Sendmail

Zurücklehnen und mein Perl Skript ausführen:

```
cd /chroot
 # die nächste Zeile unkommentieren, wenn nicht meine Konfigurationsdatei
 # verwendet wird
# ./Config_Chroot.pl config  sendmail
./Config_Chroot.pl install sendmail
./Config_Chroot.pl start   sendmail
```

Kein Probleme? Doch. Es wird immer noch als root ausgeführt. Auch werden einige Dateien immer wieder beim Start von sendmail wiederhergestellt. Mein Skript unterstützt das nicht. Jedesmal, wenn Veränderungen in /etc/mail vorgenommen werden, müssen diese auch nach /chroot/sendmail/etc kopiert werden. Weiterhin muss /var/spool/mail auf /chroot/sendmail/var/spool/mail verweisen, damit sowohl sendmail als auch die Nutzer immer die gleichen Dateien sehen.

Das gute daran ist, dass man immer Mails verschicken kann, aber mit dem Empfang gibt es Probleme. Deshalb habe ich sendmail mit apache installiert. Einige Perl Skripte versenden nun Mails und deshalb war eine Installation von sendmail in der apache Umgebung notwendig.

Andere Dinge mit chroot

Hier ist meine Philosophie:

1. alles sollte mit chroot abgesichert werden wie sendmail, ssh, apache postgresql, syslog und jeder andere auf dem Computer laufende Dienst
2. Alles sollte unter einem nicht priviligierten Account ausgeführt werden. Dazu muss man eventuell geschützte Ports auf nicht geschützte weiterleiten. (so geschehen z.B. bei sendmail und syslog)
3. Log−Dateien außerhalb sichern lassen
4. Für jeden Dienst eine eigene Partition bereitstellen, damit Cracker nicht soviel Festplattenspeicher haben, falls sie sich entscheiden, Dateien zu schreiben. Dateisysteme können ebenfalls über das Loopback Device gemountet werden.
5. Root sollte Besitzer von allen Dateien sein, die nicht geändert werden

Kommen wir jetzt noch einmal zu sendmail und syslogd. Beide sollten als nicht priviligierte User ausgeführt werden. Für sendmail sollte das möglich sein, auch wenn es sehr schwer ist, das zu konfigurieren. Ich war leider nicht erfolgreich dabei. Trotzdem halte ich es für einen schwerwiegenden Fehler. Ich weiß, es gibt da einige Probleme damit, aber ich denke, es müsste möglich sein. Solange man nämlich auf Dateiberechtigungen setzt, sehe ich keinen Grund, warum man nicht auf root Accounts verzichten kann. Dafür können natürlich Gründe existieren, die ich jetzt einfach übersehen habe, aber auch diese Gründe sollten kein unüberwindbares Hindernis darstellen.

Syslog habe ich gar nicht erst probiert, aber auch hier sehe ich keinen Grund, warum es nicht möglich sein sollte, Log−Dateien als nicht Superuser zu schreiben. Wenigstens habe ich es geschafft, syslog in abgesicherte Umgebungen zu zwingen.

Alle Dienste sollten als nicht root Accounts eingestellt werden. Sogar NFS. Wirklich alle!

Vorschläge

- Man sollte 2 Logins für ssh und zwei ssh Dämons benutzen
- Man sollte herausfinden, wie man sendmail und andere Dienste als nicht root laufen lässt.
- Man sollte die unnötigen Bibliotheken unter /lib rauswerfen. Ich habe einfach alle rauskopiert. Die meisten braucht man nämlich nicht.
- Log−Dateien sollte man auf einem entfernten System sichern und man sollte herausfinden, ob man syslog an einen Netzwerk Port binden kann. Auch sollte syslog als nicht root ausgeführt werden.

Schlussfolgerung

Ich denke, chroot ist eine wirklich coole Sache für alle Dienste. Ich denke, es ist ein großer Fehler, nicht alle Dienste abgesichert als nicht root laufen zu lassen. Hoffentlich tut dies eine der großen Distributionen, oder eine kleine oder überhaupt eine. Mandrake hatte ja RedHat als Grundlage der eigenen Distribution damals genommen und viele Dinge verbessert. Vielleicht nimmt ja jetzt jemand Mandrake und fügt chroot Unterstützung für alle Dienste hinzu. Nichts hält Leute davon ab, die Arbeit der anderen Menschen an GNU/Linux zu nutzen, deshalb müsste es möglich sein. Würde eine Firma eine komplette chroot Umgebung hinbekommen, sie würden eine fantastische Distribution haben! Man sollte immer daran denken, jetzt wo Linux Mainstream wird, dass Leute keine Kommandozeile sehen wollen. Alles sollte über eine graphische Benutzerfläche einstellbar sein und den Nutzer von unnötigen Details verschonen.

Ich unterstütze die Idee der kompletten chroot Umgebungen 100%ig, genauso wie die nicht root Accounts für die Dienste. Jede Distribution sollte dies eigentlich bieten, zumindest wenn sie im produktiven Umfeld eingesetzt werden soll.

Ich plane eine HowTo über chroot zu erstellen. Vielleicht kann mir ja jemand dabei helfen diesen Artikel in das LyX Format umzuwandeln, damit es in die Linux HowTo aufgenommen werden kann.

Hinweis

1. Falls eine Aktualisierung dieses Artikels notwendig wird, wird sie hier unter http://www.gnujobs.com/Articles/23/chroot.html verfügbar sein.