SelfLinux-0.13.1

xinetd

Autor: Florian Frank (*florian@pingos.org*)
Autor: Harald Radke (*harryrat@gmx.de*)
Autor: Frédéric Raynal (*pappy@users.sourceforge.net*)
Formatierung: Florian Frank (*florian@pingos.org*)

Das Programm `xinetd`, extended Internet services daemon, stellt einen guten Schutz gegenüber unbefugten Zugriffen dar und vermindert die Gefahr eines Denial of Service Angriffes. Ebenso, wie die bewährte Kombination von `inetd` und tcpd, steuert es die Zugriffsrechte des Rechners, leistet darüberhinaus aber noch viel mehr. In diesem Artikel werden die Fähigkeiten von `xinetd` vorgestellt.

Inhaltsverzeichnis

1 Was ist xinetd?

2 Übersetzen und Installatieren

3 Konfiguration

4 Die Sektion: default

5 Konfiguration eines Dienstes

6 Ports an Adressen binden: Das Attribut bind

7 Umleiten eines Dienstes auf einen anderen Rechner

8 Spezielle Dienste

9 chroot für einen Dienst

10 Fazit

1 Was ist xinetd?

Das klassische `inetd` dient der Steuerung und Überwachung von Netzwerkverbindungen eines Rechners. Trifft eine Anfrage auf einem Port ein, der von `inetd` verwaltet wird, wird diese an das Programm tcpd weitergeleitet. Dieses entscheidet auf Basis der Regeln in den Dateien hosts.allow und hosts.deny, ob die Anfrage zulässig ist. Ist dies der Fall, so wird der entsprechende Serverprozess , zum Beispiel ftpd, gestartet und die Anfrage bearbeitet. Dieser Mechanismus ist auch als TCP_Wrapper bekannt.

`xinetd` bietet ähnliche Möglichkeiten zur Zugriffssteuerung an, wie der TCP_Wrapper. Zusätzlich leistet es noch mehr:

* Zugriffssteuerung von TCP, UDP und RPC Diensten (letzteres funktioniert nicht ganz so gut)
* Zugriff zu bestimmten Zeiten
* Protokollierung von zulässigen, als auch von unzulässigen Anfragen
* effizienter Schutz vor Denial of Service Angriffen. Dies sind Angriffe, bei denen versucht wird, den Rechner zu blockieren, indem dessen Resourcen in großem Maße belegt werden:
  * Beschränkung der Zahl der Server des gleichen Types, die simultan laufen dürfen
  * Festlegung der maximalen Zahl gleichzeitig laufendener Server
  * Beschränkung der Größe der Protokolldateien
* Dienste können an bestimmte Netzwerkschnittstellen gebunden werden. Dadurch können einzelne Dienste etwa innerhalb eines privaten Netzwerkes verwendet werden, stehen aber nach außen hin nicht zur Verfügung.
* Verwendung als Proxy. Recht nützlich in Verbindung mit ip_masquerading (bzw. NAT), wodurch Rechner eines internen Netzwerkes von außen her erreicht werden können, obwohl nur eine IP bekannt ist

Das Hauptproblem, wie schon erwähnt, stellt die Abwicklung von RPC Aufrufen dar. Jedoch arbeiten `portmap` und `xinetd` großartig zusammen.

Im ersten Teil dieses Artikels wird erklärt, wie `xinetd` funktioniert. Ausführlich wird die Konfiguration des Systems, sowie einige spezielle Optionen (Beschränkung auf ein Netzinterface, Umleitung) beschrieben. Dabei werden zur Veranschaulichung einige Beispiele vorgestellt. Im zweiten Teil schließlich wird xinetd im laufen Betrieb betrachtet und ein Blick auf die Protokolldateien geworfen. Zum Schluß gibt es dann noch einen nützlichen Tip.

2 Übersetzen und Installatieren

Von http://www.xinetd.org/ kann man das Programm beziehen. In diesem Artikel wurde Version 2.1.8.9pre10 verwendet. Übersetzung und Installation sehen wie üblich aus:

```
root@linux / # ./configure
root@linux / # make
root@linux / # make install
```

`configure` bietet auch hier die üblichen Optionen. Desweiteren gibt es zur Übersetzungszeit drei spezielle Schalter:

1. **--with-libwrap**: Wird diese Option verwendet. so überprüft **xinetd**, ob die Konfigurationsdateien für `tcpd` (/etc/hosts.allow und /etc/hosts.deny) zu finden sind. Falls auf diese zugegriffen werden kann, verwendet `xinetd` seine Kontrolmechanismen. Soll diese Option verwendet werden, so müssen tcp_wrapper und die notwendigen Bibliotheken auf dem System installiert sein (Bemerkung: alles, was der Wrapper leistet, kann auch mit xinetd realisiert werden. Entscheidet man sich dennoch dafür, steigt die Zahl der Konfigurationsdateien unnötig an und die Administration wird aufwendiger. Es ist daher nicht zu empfehlen).
2. **--with-loadavg**: Übersetzt man `xinetd` mit dieser Option, kann später bei der Konfiguration des Programmes die Option `max_load` verwendet werden. Durch diese können Server deaktiviert werden, sollte der Rechner zu überlasten drohen, eine wesentliche Einstellung zur Abwehr von DoS Angriffen.
3. **--with-inet6**: Unterstützung für IPv6. Sowohl IPv4, als auch IPv6 Verbindungen werden verwaltet, allerdings werden ggfs. IPv4 Adressen in das IPv6 Format umgewandelt.

Es ist nicht notwendig, vor dem Start von `xinetd` das Programm `inetd` zu beenden. Dies kann aber zu einem unerwarteten Verhalten beider Programme führen!

Über folgende Signale kann man das Verhalten von `xinetd` beeinflussen:

* **SIGUSR1**: Die Konfigurationsdatei wird erneut ausgelesen und die Einstellungen für die jeweiligen Dienste werden entsprechend angepasst.
* **SIGUSR2**: Wie oben, zusätzlich werden unnötige Daemonprogramme beendet.
* **SIGTERM**: Beendet xinetd und alle Daemonprogramme, die gestartet worden waren.

Die oben erwähnten Signale können recht bequem mittels eines Skriptes verwendet werden, und z. B. über Stichworte wie start, stop, restart, soft und hard als Parameter des Skriptes gesendet werden (die letzten beiden korrespondieren zu **SIGUSR1** und **SIGUSR2**).

3 Konfiguration

Die Datei /etc/xinetd.conf dient standardmäßig der Konfiguration des `xinetd` Daemons (eine alternative Datei kann als Kommandozeilenparameter übergeben werden). Ist die Konfiguration von `xinetd` auch nicht allzu komplex, so doch ein wenig aufwendiger und die Syntax unterscheidet sich erheblich von der des Vorgängerprogrammes `inetd`.

Es gibt zwei Werkzeuge namens `itox` und `xconv.pl`, welche mit `xinetd` mitkommen und den Inhalt von /etc/inetd.conf als Basis für die Konfigurationsdatei von `xinetd` verwenden. Damit ist es das natürlich nicht gewesen, da die Regeln, definiert in der Wrapper Konfiguration, nicht berücksichtigt werden. `itox` wird wohl nicht mehr weiterentwickelt. Generell ist `xconv.pl` die bessere Wahl, auch wenn die Ergebnisse noch zusätzlich per Hand angepasst werden müssen, da `xinetd` mehr Funktionalität als `inetd` bietet.

```
root@linux / # /usr/local/sbin/xconv.pl < /etc/inetd.conf >
/etc/xinetd.conf
```

Am Anfang der Konfigurationsdatei stehen die Standardeinstellungen. Alle Attribute, die hier zu finden sind, werden für jeden Dienst, der von `xinetd` verwaltet wird, verwendet. Für jeden Dienst folgt nun ein eigener Bereich, in welchem Standardeinstellungungen angepasst werden können.

Der Eintrag für die Standardeinstellungen sieht so aus:

```
defaults
{
  attribut operator wert(e)
  ...
}
```

Die Werte jedes Attributes, die diesem hier zugwiesen werden, sind für alle folgenden Dienste gültig. Deswegen kann man mittels des Attributes `only_from` eine feste Liste von IP Adressen angeben, die die Dienste nutzen dürfen:

```
only_from = 192.168.1.0/24/192.168.5.0/24 192.168.10.17
```

Dadurch dürfen alle Rechner mit obigen Adressen alle hier konfigurierten Dienste in Anspruch nehmen. Aber natürlich können diese Werte zusätzlich noch für jeden Dienst einzeln modifiziert werden (man werfe einen Blick auf die weiter unten vorgestellten Operatoren). Dies ist allerdings ein wenig riskant. Es ist besser, will man das System einfach und sicher halten, nicht erst Standardwerte zu definieren, die dann später für einzelne Dienste wieder eingeschränkt werden. Betrachtet man beispielsweise Zugriffsrechte, so besteht die einfachste Vorgehensweise darin, zuerst generell den Zugriff zu verbieten und dann für jeden Dienst einzeln die anvisierten Benutzer anzugeben und diesen den Zugriff zu ermöglichen (bei tcp_wrapper wird dies dadurch realisiert, dass zuerst in der Datei hosts.deny die Regel **ALL:ALL@ALL** eingetragen wird und dann in hosts.allow alle zugelassenen Dienste und die erlaubten IP Adressen etwaiger Benutzer angegeben werden).

Die Syntax eines Eintrags für einen Dienst in xinetd.conf sie folgendermaßen aus:

```
service
{
```

```
  attribut operator wert(e)
  ...
}
```

Als Operatoren stehen zur Auswahl: =, += und -=. Die meisten Attribute lassen jedoch nur den Operator = zu. Dieser weist einem Attribut einen bestimmten Wert zu. Durch += wird ein Eintrag einer Liste von Werten hinzugefügt, mittels -= hingegen wird der Eintrag aus der Liste entfernt.

Im Folgenden gibt es eine kurze Übersicht der Attribute. Deren Einsatz wird in einigen Beispielen verdeutlicht werden. Weitergehende Informationen kann man der Manpage zu `xinetd.conf` entnehmen.

Attribut	**Werte und Beschreibung**
Flags	Nur die gängigsten Werte werden hier aufgeführt; weitere sind in der Dokumentation zu finden: * **IDONLY**: Es werden nur Verbindungen mit Client-Rechnern zugelassen, auf denen ein Server zur Identifikation läuft. * **NORETRY**: Unterbindet das Anlegen eines neuen Prozesses im Fehlerfall. * **NAMEINARGS**: Das erste Argument des Attributes server_args wird als argv[0] des Servers verwendet. Dadurch wird es möglich, tcpd zu verwenden, indem es dem Attribut server zugewiesen wird, gefolgt vom Namen des Servers und dessen Parameter als sever_args, genau, wie bei `inetd`.
log_type	`xinetd` verwendet syslog, als Selektor wird standardmäßig `daemon.info` gewählt. * SYSLOG Selektor [level]: Auswahl zwischen `daemon`, `auth`, `user` oder `local0-7` des `syslog` Daemons. * FILE [max_size [absolute_max_size]]: In die hier angegebene Datei wird die Ausgabe gesichert. Die beiden Optionen begrenzen die Dateigröße. Wird der erste Wert erreicht, wird eine Nachricht an syslog gesendet. Die Protokollierung des jeweiligen Dienstes wird eingestellt, sobald die Dateigröße den zweiten Parameter erreicht (sollte es sich um eine mehrfach verwandte Datei oder um eine Standardeinstellung in `xinetd.conf` handeln, können auch mehrere Dienste betroffen sein).
log_on_success	Bei Start eines Servers können mehrere unterschiedliche Informationen festgehalten werden: * **PID**: die Server PID (handelt es sich um einen internen Dienst von `xinetd` ist die PID 0) * **HOST**: Die Adresse des Clients * **USERID**: Die ID des Benutzers, der den Dienst in Anspruch nimmt, entsprechend dem in der RFC1413 definierten Protokolles * **EXIT**: Der Statuswert, den der Prozess bei Beendigung zurückliefert * **DURATION**: Verbindungsdauer
log_on_failure	Hierdurch kann eine Vielzahl an Informationen protokolliert werden, sobald ein Server nicht gestartet werden kann, sei es aufgrund fehlender Resourcen oder der Verletzung der Zugriffsregeln: * **HOST, USERID**: wie oben * **ATTEMPT**: Hält einen Zugriffsversuch fest. Diese Option wird automatisch gewählt, sobald ein weiterer Wert gesetzt wird * **RECORD**: Protokolliert alle Informationen, die über den Client verfügbar sind.
nice	Ändert die Priorität des Servers, analog dem Unixbefehl `nice`.
no_access	Liste der Clients, denen der Zugriff auf den jeweiligen Dienst verweigert werden soll
only_from	Liste aller akzeptierten Rechner. Wird diesem Attribut kein Wert zugewiesen, so ist kein

	Zugriff auf den Dienst möglich
port	Port, unter dem der Dienst erreichbar ist. Sollte dieser auch in der Datei /etc/services stehen, müssen beide Wert übereinstimmen.
protocol	Das hier angegebene Protokoll muss in der Datei /etc/protocols aufgeführt sein. Wird dieses Attribut nicht gesetzt, wird das Standardprotokoll des Servers verwendet
server	Pfad, unter dem der Server zu finden ist
server_args	Parameter, die an den Server übergeben werden sollen
socket_type	stream (TCP), dgram (UDP), raw (IP Direktzugriff) oder seqpacket ()
type	`xinetd` verwaltet drei Arten von Diensten: 1. **RPC**: alle, die in der Datei /etc/rpc zu finden sind. Funktioniert nicht besonders gut 2. **INTERNAL**: Dienste, die direkt von `xinetd` verwaltet werden (echo, time, daytime, chargen und discard) 3. **UNLISTED**: Dienste, die weder in /etc/rpc noch in /etc/services stehen Bemerkung: man kann mehrere Werte kombinieren, wie bei den internen Diensten servers, services und xadmin gezeigt wird.
wait	Legt das threading-Verhalten des Dienstes fest. Es gibt zwei Möglichkeiten: * **yes**: Der Dienst ist mono-threaded, dass heisst es kann immer nur eine Verbindungen dieses Types unterhalten werden * **no**: Für jede neue Anfrage startet xinetd einen neuen Serverprozess, unter Berücksichtigung etwaiger Beschränkungen der maximalen Zahl an Servern (Achtung: Standardmäßig gibt es keine Obergrenze für die Serverzahl)
cps	Beschränkt die Zahl der zugelassenen Verbindungen. Als erster Parameter wird die Zahl selbst angeben. Der zweite Wert legt die Zeit fest, die der Dienst für weitere Verbindungen erreichbar ist, sobald das Limit erreicht wurde.
instances	Maximale Zahl an Servern des gleichen Types, die simultan laufen dürfen
max_load	Tatsächliche maximale Last eines Servers (z. B. 2 oder 2.5). Wird dieser Wert überschritten, nimmt der Server keine weiteren Anfragen an
per_source	Ein Zahlenwert oder UNLIMITED. Ermöglicht es, die Zahl der Verbindungen, die ein Dienst mit einem einzelnen Clientrechner unterhält, zu beschränken

Die letzten vier Attribute bieten die Möglichkeit, die Resourcen je nach Server zu verwalten. Dies stellt einen effizienten Schutz vor Denial of Service Angriffen dar.

In diesem Abschnitt wurden ein wenig von der Funktionalität von `xinetd` vorgestellt. Im nächsten wird nun gezeigt, wie diese angewandt wird und einige Regeln erläutert, so dass xinetd korrekt funktioniert.

4 Die Sektion: default

In diesem Abschnitt können Standardwerte für eine Anzahl von Attributen (eine komplette Liste ist in der Dokumentation zu finden) gesetzt werden. Einige der Attribute, wie etwa `only_from`, `no_access`, `log_on_success` oder `log_on_failure`, können gleichzeitig Werte dieser Sektion und der der einzelnen Dienste beinhalten.

Der erste Schritt hin zu einem sicheren System ist, standardmäßig allen Rechnern den Zugriff zu verweigern. Danach wird dann für den jeweiligen Dienst getrennt festgelegt, wer diesen nutzen darf. Wie gesehen gibt es zwei Attribute, die den Zugriff auf Basis der IP Adresse eines Rechners steuern: `only_from` und `no_access`. Unter Verwendung des letzteren:

```
no_access = 0.0.0.0/0
```

wird der Zugriff auf alle Dienste unterbunden. Soll dann etwa **echo** allen zugängig gemacht werden, sollte in der Sektion für den **echo** Dienst folgendes stehen:

```
only_from = 0.0.0.0/0
```

Unter Verwendung dieser Einstellungen erhält man folgende Ausgabe:

```
Sep 17 15:11:12 charly xinetd[26686]: Service=echo-stream: only_from
list and no_access list match equally the address 192.168.1.1
```

Bei der Entscheidung, ob ein Zugriff erlaubt werden soll, werden die Adresslisten beider Attribute überprüft. Wird die Adresse des Clientrechners von beiden Listen überdeckt, so ist das Attribut relevant, welches weniger allgemein ist. Kann dies nicht festgestellt werden (beide gleich allgemein), so wird die Verbindung nicht zugelassen. Um dies zu beheben, sollte man stattdessen folgendes schreiben:

```
only_from = 192.0.0.0/8
```

Eine einfachere Lösung besteht darin, in der Sektion defaults folgendes Attribut zu verwenden:

```
only_from =
```

Wird kein Wert zugewiesen, scheitert jeder Verbindungsaufbau. Danach kann jeder Dienst den Zugriff über eine entsprechende Zuweisung an dieses Attribut regeln.

Wichtig: Wird gar keine Zugriffsregel (also weder das Attribut `only_from`, noch `no_access`) für einen Dienst aufgestellt (weder in dessen, noch in der defaults Sektion), so kann jeder diesen Dienst nutzen!

Ein Beispiel für die defaults Sektion:

```
defaults
{
  instances       = 15
  log_type        = FILE /var/log/servicelog
  log_on_success  = HOST PID USERID DURATION EXIT
  log_on_failure  = HOST USERID RECORD
  only_from       =
  per_source      = 5

  disabled = shell login exec comsat
  disabled = telnet ftp
  disabled = name uucp tftp
  disabled = finger systat netstat

  #INTERNAL
  disabled = time daytime chargen servers services xadmin

  #RPC
  disabled = rstatd rquotad rusersd sprayd walld
}
```

Mittels servers, services und xadmin unter #INTERNAL kann `xinetd` gesteuert werden. Später dazu mehr.

5 Konfiguration eines Dienstes

Was muss man nun noch zusätzlich tun, um einen Dienst zu konfigurieren? Es müssen noch gegebenenfalls die Standardwerte einiger Attribute angepasst bzw. weitere Attribute für den jeweiligen Dienst verwendet werden.

Einige Attribute müssen, je nach Art des Dienstes (INTERNAL, UNLISTED oder RPC), angegeben werden:

Attribut	Verwendung
socket-type	Jeder Dienst
user	Nur Dienste, die keine INTERNAL Dienste sind
server	Nur Dienste, die keine INTERNAL Dienste sind
wait	Alle Dienste
protocol	Alle RPC Dienste und solche, die nicht in /etc/services zu finden sind
rpc_version	Jeder RPC Dienst.
rpc_number	Jeder RPC Dienst ohne Eintrag in /etc/rpc.
port	Jeder Dienst, der kein RPC Dienst ist und nicht in /etc/services steht

Hier ein Beispiel, wie die Konfiguration von Diensten aussieht:

```
service ntalk
{
  socket_type   = dgram
  wait          = yes
  user          = nobody
  server        = /usr/sbin/in.ntalkd
  only_from     = 192.168.1.0/24
}

service ftp
{
  socket_type  = stream
  wait         = no
  user         = root
  server       = /usr/sbin/in.ftpd
  server_args  = -l
  instances    = 4
  access_times = 7:00-12:30 13:30-21:00
  nice         = 10
  only_from    = 192.168.1.0/24
}
```

Man beachte, dass diese Dienste nur im lokalen Netzwerk zugäglich sind (293.268.1.0/24). Was FTP angeht, so wurde der Zugriff noch weiter eingeschränkt: Nur 4 Rechnern wurde der Zugriff gestattet und dies auch nur zu bestimmten Zeiten.

6 Ports an Adressen binden: Das Attribut bind

Mittels diesem Attribut kann man einen Dienst an eine IP Adresse binden. Besitzt der Rechner sowieso nur eine Adresse, ist dies nicht sehr sinnig. Ein Rechner allerdings, der einerseits an einem lokalen Netzwerk, andereseits an das Internet angebunden ist, besitzt mindestens zwei Adressen.

Angenommen, eine Firma möchte einen FTP Server für ihre Angestellten installieren, etwa, damit diese auf interne Dokumente zugreifen und diese lesen können. Desweiteren sollen Kunden mittels FTP auf die Firmenprodukte zugreifen können. `bind` ist wie geschaffen für diesen Fall. Es werden zwei FTP Dienste definiert. Diese müssen allerdings von `xinetd` unterschieden werden können. Dazu wird das Attribut `id` verwendet. Durch dieses wird ein Dienst eindeutig identifiziert. Sollte ein Dienst dem Attribut keinen Wert zuweisen, so wird standardmäßig der Dienstname als Wert verwendet.

```
service ftp
{
  id           = ftp-public
  wait         = no
  user         = root
  server       = /usr/sbin/in.ftpd
  server_args  = -l
  instances    = 4
  nice         = 10
  only_from    = 0.0.0.0/0 #allows every client
  bind         = 212.198.253.142 #public IP address for this server
}

service ftp
{
  id           = ftp-private
  socket_type  = stream
  wait         = no
  user         = root
  server       = /usr/sbin/in.ftpd
  server_args  = -l
  only_from    = 192.168.1.0/24 #only for internal use
  bind         = 192.168.1.1  #local IP address for this server (charly)
}
```

Durch den Einsatz von `bind` kann das jeweilige Daemonprogramm angesprochen werden, abhängig von der Zieladresse der Pakete. Will ein Client im lokalen Netzwerk auf interne Daten zugreifen, so muss er die lokale Adresse (bzw. den assoziierten Rechnernamen) des FTP Servers verwenden. In der Protokolldatei steht dann:

```
00/9/17@16:47:46: START: ftp-public pid=26861 from=212.198.253.142
00/9/17@16:47:46: EXIT: ftp-public status=0 pid=26861 duration=30(sec)
00/9/17@16:48:19: START: ftp-internal pid=26864 from=192.168.1.1
00/9/17@16:48:19: EXIT: ftp-internal status=0 pid=26864 duration=15(sec)
```

Der erste Teil stammt vom Befehl `ftp 212.198.253.142`, während der zweite Teil vom Aufruf von charly auf sich selbst resultiert: `ftp 192.168.1.1`.

Was passiert aber nun, falls ein Rechner keine zwei statischen IP Adressen hat? Dies ist zum Beispiel bei PPP Verbindungen oder beim Einsatz von DHCP der Fall. Es wäre besser, könnten Dienste an Netzwerkschnittstellen selbst, anstatt an Adressen gebunden werden. Dies jedoch ist bei `xinetd` noch nicht in Sicht und stellt ein wirkliches Problem dar. Das Design eines C Modules beispielsweise, dass auf ein Netzwerkinterface oder eine

Adresse zugreifen soll, ist stark vom jeweiligen Betriebssystem abhängig, und da `xinetd` auf vielen Plattformen arbeiten soll, ist dies problematisch. Unter Verwendung eines Skriptes kann das Problem gelöst werden:

```
#!/bin/sh

PUBLIC_ADDRESS=`/sbin/ifconfig $1 | grep "inet addr" | awk '{print $2}'| awk -F: '{print
$2}'`
sed s/PUBLIC_ADDRESS/"$PUBLIC_ADDRESS"/g /etc/xinetd.base > /etc/xinetd.conf
```

Dieses Skript liest die gewünschte Konfiguration aus der Datei /etc/xinetd.base aus, wobei **PUBLIC_ADDRESS** als Platzhalter für die dynamische Adresse steht und nimmt die entsprechenden Änderungen in /etc/xinetd.conf vor. Hierbei wird **PUBLIC_ADDRESS** durch die Adresse ersetzt, die dem Interface zugeordnet ist, welches dem Skript als Parameter übergeben wird. Der Aufruf des Skriptes hängt von der Art der Verbindung ab: Am einfachsten ist es, den Aufruf in die jeweilige ifup-* Datei einzutragen und `xinetd` neu zu starten.

7 Umleiten eines Dienstes auf einen anderen Rechner

`xinetd` kann auch als ein transparenter Proxy verwendet werden (jedenfalls beinahe, wie später zu sehen sein wird) und zwar mittels dem Attribut `redirect`. Dieses erlaubt es, eine Anfrage an einen Dienst an einen anderen Rechner und einem bestimmten Port weiterzuleiten.

```
telnet service
{
  flags  = REUSE
  socket_type = stream
  wait  = no
  user  = root
  server = /usr/sbin/in.telnetd
  only_from = 192.168.1.0/24
  redirect = 192.168.1.15 23
}
```

Dabei passiert folgendes:

```
user@linux / $ telnet charly

Trying 192.168.1.1...
Connected to charly.
Escape character is '^]'.

Digital UNIX (sabrina) (ttyp1)

login:
```

Es scheint, als ob eine Verbindung mit dem Rechner charly aufgebaut worden ist. Tatsächlich aber befindet sich sabrina auf der anderen Seite der Verbindung (eine Alpha Maschine, auch an **Digital UNIX** zu erkennen). Die Weiterleitung kann recht nützlich, aber auch gefährlich sein.

8 Spezielle Dienste

Es gibt drei Dienste, die nur für `xinetd` verwendet werden. Diese sind weder in /etc/rpc, noch in /etc/services zu finden und müssen das UNLISTED Flag setzen (neben dem INTERNAL Flag, welches anzeigt, dass sie `xinetd` Dienste sind).

1. **servers**: Auskunft über die gerade laufenden Server
2. **services**: liefert Informationen über die angebotenen Dienste, ihre Protokolle und Ports
3. **xadmin**: bietet Funktionen der beiden oben genannten Dienste

Es ist offensichtlich, dass durch diese Dienste das System verletzbarer wird. Sie stellen wichtige Informationen zu Verfügung. Zur Zeit gibt es keine Sicherheitsmechanismen für den Zugriff auf diese Dienste (etwa Passwortschutz). Sie sollten nur in der Einrichtungsphase verwendet werden. Zunächst wird der Zugriff auf sie in dem Abschnitt defaults generell gesperrt:

```
defaults {
  ...
  disabled = servers services xadmin
  ...
}
```

Bevor sie dann aktiviert werden sollten einige Vorkehrungen getroffen werden:

1. Der Rechner, auf dem `xinetd` läuft, sollte der einzige Rechner sein, dem es erlaubt ist, die Dienste zu nutzen
2. Die Anzahl der gleichzeitig erlaubten Serverinstanzen sollte eins sein
3. Zugegriffen werden darf nur von dem Rechner aus, auf dem der Server läuft.

Hier ein Beispiel für die Konfiguration des **xadmin** Dienstes. Die Einstellung der beiden anderen Dienste erfolgt analog, bis auf die Portnummer natürlich.

```
service xadmin
{
  type  = INTERNAL UNLISTED
  port  = 9100
  protocol = tcp
  socket_type = stream
  wait  = no
  instances = 1
  only_from = 192.168.1.1  #charly
}
```

Der Dienst xadmin bietet fünf Befehle:

1. help ...
2. show run : gleicht dem Dienst servers und gibt Auskunft über die zur Zeit laufenden Server
3. show avail : gleicht dem Dienst services und informiert über die verfügbaren Dienste (und ein wenig mehr)
4. bye oder exit ...

Das System kann auch ohne diese Dienste getestet werden. Befehle, wie `netstat`, `fusert`, `lsof` geben ebenfalls einen Überblick über das, was auf dem Rechner vonstatten geht, ohne ihn ein wenig unsicherer durch

die Verwendung der Dienste zu machen!

9 chroot für einen Dienst

Oft wird empfohlen, die Bereiche eines Dienstes einzuschränken oder eine neue Umgebung zu generieren. Der Befehl `chroot` ändert das Wurzelverzeichnis für einen Befehl oder ein Skript:

```
root@linux / # chroot [options] new_root
```

Auf diese Weise werden häufig Dienste wie bind/DNS oder FTP gesichert. Um dies zu erreichen und dabei gleichzeitig die Vorzüge von `xinetd` nutzen zu können, muss `chroot` als Server deklariert werden. Die Parameter werden dann über das Attribut server_args übergeben.

```
service ftp
{
  id           = ftp
  socket_type  = stream
  wait         = no
  user         = root
  server       = /usr/sbin/chroot
  server_args  = /var/servers/ftp /usr/sbin/in.ftpd -l
}
```

Sobald nun eine Anfrage, den jeweiligen Dienst betreffend, eintrifft, wird als erstes `chroot` ausgeführt. Das erste Argument, dass diesem übergeben wird, ist der erste Eintrag in sever_args: das neue Wurzelverzeichnis. Zuletzt wird der Server gestartet.

10 Fazit

Es mag sich nun die Frage stellen, ob nun `xinetd` oder `inetd` verwendet werden sollte. Es ist nun einmal so, dass `xinetd` ein wenig mehr Administration verlangt, vorallem, solange es nicht mit den Linux Distributionen mitgeliefert wird. Eine sichere Lösung stellt der Einsatz von `xinetd` auf Rechnern dar, die eine Verbindung mit dem Rest der Welt (etwa dem Internet) haben. `xinetd` bietet einfach den besseren Schutz. Für Rechner in einem lokalen Netzwerk sollte `inetd` vollkommen ausreichend sein.