Berufliche Schulen Bretten
Klasse: TGI 13/1
Fach: Computertechnik
Thema: Betriebssysteme und vernetzte Systeme
Lehrer: Herr Körner
Schuljahr: 2008/2009

Entwicklung und Implementierung einer modernen modularen, netzwerkbasierten Backup-Lösung

Ausarbeitung

vorgelegt von: **Dennis Felsing**

Beethovenstraße 6b
75056 Sulzfeld
E-Mail: dennis@felsin9.de

2009-04-26

Inhaltsverzeichnis

Kapitel 1

Einleitung

Egal ob kleines oder großes Unternehmen, Forschungsinstitut, oder Privatperson, nahezu jeder legt auf dem Datenspeicher seiner Computer Daten an, die eine hohe Bedeutung haben. Es kann sich dabei beispielsweise um die Photosammlung vom letzten Familienurlaub, wichtige Kundendaten, eine Webpublikation, oder das gesamte System handeln. Unabhängig von ihrem objektiven Nutzen fordert der Verlust oder die Beschädigung dieser Daten für die Benutzer des Gerätes einen oder mehrere zum Beispiel dieser Preise: Arbeitszeit, Kapital, persönlicher Schaden. Aus diesen Gründen sind die Daten vor den Gefahren, denen sie ausgesetzt sind, zu schützen. Diese Gefahren schließen fehlerhafte und Schadsoftware, böswillige und unvorsichtige Benutzer, und Hardwareschäden ein.

Schon früh hat man ausgeklügelte Programme entworfen, um diesen Gefahren und ihren Auswirkungen entgegenzuwirken. In den 1970-ern entstand eine Software namens *tar*, was für tape archive steht, und die dazu dient mehrere Dateien auf Band zu sichern. Insbesondere nützlich für Backups ist die Möglichkeit differenzielle und inkrementielle Archive zu erstellen. Es war eine Möglichkeit geschaffen Daten zuverlässig und platzsparend zu sichern. Bei der Wiederherstellung lässt sich jedoch nicht so einfach eine einzelne Datei vom Band zurückspielen, sollte nur diese beschädigt sein. Dennoch ist dies die traditionelle Methode zur professionellen Erstellung von Backups, und sie wird auch heute noch weitläufig eingesetzt, insbesondere mit Hilfe von *tar* und *dump*, einem von BSD stammenden Tool, das ähnlich wie *tar*, jedoch auf Dateisystemebene, arbeitet.

Ich möchte eine fortschrittlichere Möglichkeit zur Erstellung von Backups vorstellen, die in der Nutzung und bei der Wiederherstellung komfortabler als die Sicherung auf Bänder ist. Es handelt sich dabei lediglich um eine einfache Verknüpfung einzelner Programme, die jeweils ihre Aufgabe hervorragend lösen können und durch diese Verknüpfung einen komplett neuen Nutzen erfahren. Diese Lösung will ich anhand meiner Heimnetzwerke erläutern, für die ich sie entwickelt und in denen ich es eingerichtet habe.

Kapitel 2

Konzept

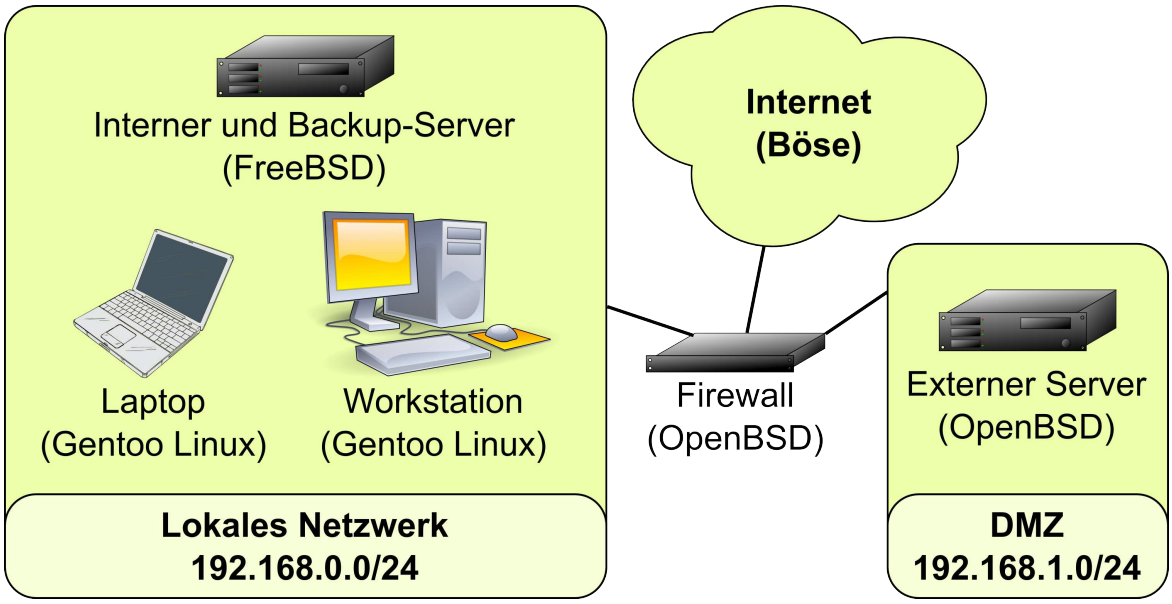

Abbildung 2.1: Schematischer Netzwerkaufbau mit Netzwerken und einigen Netzwerkgeräten

Um die Rechner des Lokalen Netzwerks (englisch: Local Area Network, LAN) vor einem Angriff aus dem Internet zu schützen, befinden sich alle von außen erreichbaren Dienste auf einem seperaten Server, der sich in einer Entmilitarisierten Zone (englisch: Demilitarized Zone, DMZ) befindet. Sind in den dort laufenen Diensten Sicherheitslücken vorhanden, die einem Angreifer Zugriff auf den Server ermöglichen, so hat dieser keine Möglichkeit auf die anderen Computer zuzugreifen, da die Firewall alle Zugriffe aus der DMZ in das LAN verhindert. Dies hat jedoch den ungewollten Nebeneffekt, dass der externe Server nicht eigentständig Backups auf den Backup-Server machen kann.

Die hier vorgestellte Backup-Lösung verfährt stattdessen so, dass der Backup-Server den Backup-Prozess einleitet. Dazu benötigt er root-Zugriff per SSH[1] auf alle zu sichernden Rechner. Auf den Backup-Clients werden lediglich ein laufender *sshd* (SSH

[1]SSH: Secure Shell

Daemon), der den Zugriff vom Backup-Server erlaubt, und *rsync* benötigt. Desweiteren muss der Server die Berechtigung haben sich mit seinem Schlüssel anstatt eines Passworts einzuloggen. Da der Backup-Server auf den externen Server zugreift, anstatt umgekehrt, leitet die Firewall die Verbindung weiter und die Sicherung der Daten ist möglich.

Kapitel 3

Sicherung

3.1 Dateisystem-Struktur

Jeder Client hat auf dem Backup-Server einen eigenen Ordner, in dem all seine Backups gespeichert werden. Jedes Backup wird dabei mit einem Ordner bezeichnet, der als Name das Datum und die Uhrzeit der Erstellung der Sicherung enthält. Das sieht dann für den Client *exxt* wie folgt aus:

Listing 3.1: Backup-Ordner für Client *exxt*

```
1 # tree -L 1 exxt
2 exxt
3 |-- 2009-02-18T04:30:01
4 |-- 2009-02-19T04:30:01
5 |-- 2009-02-20T04:30:01
6 |-- 2009-02-21T04:30:01
7 |-- 2009-02-22T04:30:01
8 |-- 2009-02-23T04:30:01
9 |-- 2009-02-24T04:30:01
10 |-- 2009-02-25T04:30:02
11 |-- 2009-02-26T04:30:01
12 |-- 2009-02-27T04:30:02
13 `-- latest -> 2009-02-27T04:30:02
14
15 14 directories, 0 files
```

Die Backups werden allem Anschein nach periodisch automatisch angefertigt, mehr dazu im Kapitel Automatische Sicherung. Desweiteren fällt auf, dass eine weitere Datei namens *latest* existiert. Hierbei handelt es sich um eine symbolische Verknüpfung auf das letzte erstellte Backup.

Wäre gerade ein Backup im Gange wäre noch ein zusätzlicher Ordner zu sehen: *.tmp*. Er verhindert, dass nach einem Absturz des Programmes ein unvollständiges Backup für ein vollständiges gehalten wird. Erst nach dem Abschluss der Sicherung wird er umbenannt in das bereits vorgestellte Namensschema. Ein weiterer Nutzen des *.tmp*-Ordners ist die Zeitersparnis, da man mit ihm ein unvollständiges Backup zu einem späteren Zeitpunkt fortführen kann.

Neben diesem Ordner legt *rrb* (englisch: reverse rsync backup), das von mir entworfene Backup-Script, noch eine *.lock*-Datei an. Sie verhindert das mehrfache Ausführen von

rrb, da ansonsten mit fehlerhaften Backups zu rechnen wäre.

3.2 Konfiguration

3.2.1 config.hostname

Die *config.hostname* ist die Hauptkonfigurationsdatei von *rrb*. Für jeden zu sichernden Client muss auf dem Server eine solche Datei angelegt werden. Sie muss beim Ausführen von *rrb* stets angegeben werden. Hier der äußerst simple Aufbau für den Client *exxt*:

Listing 3.2: Konfiguration für *exxt*

```
1 # cat /etc/rrb/config.exxt
2 SRC=exxt:/
3 DEST_DIR=/media/backup/exxt
4 RSYNC_OPTS=( --rsync-path='nice -n19 rsync' )
5 EXCLUDES_FILE=/etc/rrb/excludes.exxt
```

Die Variable *SRC* beschreibt den Clientnamen (*exxt*) oder seine IP-Adresse und den zu sichernden Ordner (*/*). *DEST_DIR* steht für den lokalen Ordner auf dem Server, in den die Backups gesichert werden sollen. Die *RSYNC_OPTS* ermöglichen eine Anpassung des Aufrufs von *rsync*. In diesem Fall wird dem auf dem Client laufende Teil von *rsync* eine möglichst geringe Priorität zugewiesen, damit andere Programme auf dem Client nicht in ihrer Arbeit beeinträchtigt werden und die Benutzer von *exxt* keine Verzögerungen feststellen. Es gibt noch einige weitere Optionen, die unter config.example aufgelistet sind.

Die letzte Zeile stellt mit der Option *EXCLUDES_FILE* eine Datei ein, in der Dateien aufgelistet sind, die sich zwar im zu sichernden Ordner befinden, aber dennoch vom Backup ausgeschlossen werden sollen. Jetzt zum Aufbau dieser Excludes-Datei.

3.2.2 excludes.hostname

In jeder Zeile wird eine Datei[1] angegeben, die ausgeschlossen werden soll. Anstatt direkt den Dateinamen anzugeben, lässt sich mit Mustern eine fortgeschrittenere Selektion vornehmen. Eine Zeile mit dem Inhalt */home/*foo* würde beispielsweise die Ordner */home/barfoo* und */home/foofoo* ausschließen, nicht jedoch */home/foobar*. Die Angabe von *.git* würde nicht, wie man vielleicht vermuten mag, nur den Ordner */.git* ausschließen, sondern jeden, der diesen Namen trägt, also auch */home/foobar/.git*.

[1] Ordner, symbolische Links, Blockdateien, usw. sind auch Dateien.

3.3 Ablauf

Der Backup-Server lädt die zu sichernden Dateien vom Client über *rsync*. Ist bereits ein Backup von diesem Client in seinem Ordner vorhanden, so wird dieses als Basis verwendet.[2] Das bedeutet, dass unveränderte Dateien nicht neu übertragen, sondern aus dem letzten Backup-Ordner übernommen werden. Sie werden dazu jedoch nicht kopiert, sondern hart verlinkt. Dadurch muss die selbe Datei nicht noch mal auf dem Dateisystem gespeichert werden. Die vorhandenen Dateieinträge in den Ordnern zeigen auf die selbe Inode.

Rsync übertragt nur die Dateien über das Netzwerk, die seit dem letzten Backup modifiziert wurden. Dieses Konzept ähnelt der inkrementiellen Sicherung, hat jedoch einen gravierenden Unterschied: Man kann die einzelnen Backup-Stufen als vollständige Backups ansprechen. Jedes einzelne Backup liegt für den Nutzer scheinbar als vollständiges Backup vor, auf dem Dateisystem sind die identischen Dateien jedoch nur ein mal gespeichert. Lediglich die Veränderungen werden in neue Dateien gesichert. Dies vereinfacht die Wiederherstellung von Daten aus beliebigen Zeitpunkten, wie wir im Kapitel Wiederherstellung sehen werden.

3.4 Automatisierte Sicherung

Viele Server laufen permanent. In solch einem Fall bietet es sich an, täglich ein Backup zur selben Uhrzeit anzulegen. Mit *cron* ist dies in ein leichtes:

Listing 3.3: *cron*-Eintrag für *exxt*

```
1 # crontab -l
2 30 4 * * * /usr/local/bin/rrb /etc/rrb/config.exxt
```

Die Zeile besagt, dass jeden Tag um 4:30 ein neues Backup von *exxt* angelegt wird. Somit stehen stets aktuelle Backups zur Verfügung, sollte der Rechner ausfallen. Leider ist die Automatisierung eines Backups nicht immer so einfach möglich, insbesondere nicht, wenn man Rechner sichern will, die nicht 7 Tage die Woche, 24 Stunden am Tag laufen, wie eine Workstation oder ein Laptop.

Aus diesem Grund habe ich *rrb_interval* geschrieben. Dieses kleine Script erstellt nur ein Backup, wenn das letzte ein bestimmtes Alter vorweisen kann. Ansonsten wird gar nichts getan. Um es in Verbindung mit *cron* zu verwenden, kann man wie folgt vorgehen:

Listing 3.4: *cron*-Eintrag für *x*

```
1 # crontab -l
```

[2] Genauer gesagt wird die zuvor erwähnte latest-Verknpüfung als Basis verwendet.

```
2 */10 * * * * /usr/local/bin/rrb_interval 86400 /etc/rrb/config.x
```

Damit wird alle zehn Minuten *rrb_interval* aufgerufen, dass dann schaut ob das letzte Backup für den Rechner x bereits mehr als 86400 Sekunden, also genau 24 Stunden, zurückliegt. Nur dann wird ein neues Backup erstellt. Es wird somit sichergestellt, dass auch die unregelmäßig laufenden Computer gesichert werden, ohne dass der Anwender sich darum kümmern müsste.

3.5 Interaktive Sicherung

Eine Alternative oder Ergänzung zu der eben genannten Lösung wäre ein Script, das sxc[3] verwendet um Backup-Anfragen von Clients entgegenzunehmen. Damit könnte der Client entscheiden, wann seine Daten gesichert werden sollen, beispielsweise vor einer kritischen Änderung, wie einem Systemupdate oder einem Festplattentausch.

Zu diesem Zweck wurde *sxc-rrb* erstellt. Nachdem es in der *sxc-rrb-config* konfiguriert wurde, lässt es sich einfach auf dem Backup-Server ausführen. Die eingetragenen JIDs können nun per XMPP mit dem Backup-Server kommunizieren und Backups erstellen.

Listing 3.5: Nutzung von *sxc-rrb* auf dem Server per *sxc* auf dem Client

```
1 <dennis> hi backup-server
2 <backup> Invalid usage, see :help
3 <dennis> :help
4 <backup> Commands:
5            :backup name
6            :list
7 <dennis> :list
8 <backup> Valid hosts: x l m
9 <dennis> :backup x
10 <backup> Running: rrb "/etc/rrb/config.x"
11 <backup> ssh: connect to host x port 22: No route to host
12          rsync: connection unexpectedly closed (0 bytes received so far) [receiver]
13          rsync error: unexplained error (code 255) at io.c(632) [receiver=3.0.4]
14 <backup> rrb "/etc/rrb/config.x" finished with exit code: 1
15 <dennis> :backup l
16 <backup> Running: rrb "/etc/rrb/config.l"
17 <backup> receiving incremental file list
18          [ Meldungen über übertragene Dateien aus Platzgründen entfernt ]
19          sent 13471 bytes  received 3758861 bytes  42148.96 bytes/sec
20          total size is 4351643914  speedup is 1153.57
21 <backup> rrb "/etc/rrb/config.l" finished with exit code: 0
```

[3] sxc: simple xmpp client

3.6 Sicherheit

Nachdem jetzt die Sicherung von Daten funktioniert, machen wir uns einige Gedanken über die Sicherheit, die diesem Ansatz innewohnt und worauf man für einen sicheren Betrieb achten muss.

Die Verbindung zwischen zu sicherndem Client und Backup-Server findet grundsätzlich über eine durch SSH gesicherte Verbindung statt. Dadurch kann ein Man-In-The-Middle keine Daten mitlesen oder injizieren. Somit wäre selbst eine Sicherung von Computern, die über das Internet verbunden sind, möglich, was bei vielen anderen Lösungen, die auf unverschlüsselte Datentransfers vertrauen, für Client und Server gefährlich wäre.

In der Regel wird SSH mit einer Passwort-basierten Anmeldung genutzt. Für einen Backup-Server wäre dies jedoch unmöglich, da man nicht ständig die Passwörter der Clients auf dem Server eingeben will. Stattdessen legt man auf dem Backup-Server einen asymmetrischen Authentifizierungs-Schlüssel an. Der öffentliche Teil dieses Schlüssels wird bei den root-Accounts der Clients als berechtigt eingetragen. Nun kann der Backup-Server ohne Angabe eines Passworts auf seine Clients zugreifen.

Der Server hat also völligen Zugriff auf alle Clients (und selbstverständlich deren Backups). Man muss deshalb den Rechner entsprechend absichern, damit kein fremder Zugriff möglich ist. Hierzu dient (unter anderem) die DMZ, da somit kein Zugriff aus dem Internet oder von aus dem Internet befallenen Rechnern mehr auf den Backup-Server möglich ist. Außerdem soll auf dem Backup-Server nur ein Minimium an Diensten in das interne Netzwerk geöffnet werden, nämlich genau so viele, dass eine Möglichkeit zur Wiederherstellung der Daten gegeben ist.

Kapitel 4

Wartung

Mittlererweile können Backups erstellt werden. In diesem Kapitel befasse ich mich mit der Wartung der erstellten Backups.

4.1 Doppelte Dateien

Wie bereits in der Sektion Ablauf erläutert, werden bei jedem Backup-Vorgang nur die veränderten Dateien übertragen und auf der Festplatte gespeichert. Dadurch spart man schon sehr viel Speicherplatz. Es ist zusätzlich noch möglich die vorhandenen Backups nach identischen Dateien zu durchsuchen. Wenn man dabei nicht nur die Backups einer Maschine, sondern die aller berücksichtigt, lässt sich nochmals ein enormer Speicherplatzgewinn verzeichnen.

Werden beispielsweise zwei auf Windows XP basierende Systeme gesichert, so lassen sich die meisten zu Windows gehörenden Dateien hart verlinken.

Es existiert eine Vielzahl an Programmen, die dies ermöglichen, wie *hardlink*, *hardlink++*, *hardlink.py*, *duff*, *fdupes*, *dupmerge*, *fdf*, *liten*, oder *rdfind*, das ich verwende:

Listing 4.1: Aufruf von *rdfind*

```
1 # nohup nice -n19 rdfind --makehardlinks true /media/backup &
```

Mit diesem Befehl wird *rdfind* im Hintergrund (*&*) und losgekoppelt von der Shell (*nohup*) gestartet und erstellt Hardlinks für identische Dateien in */media/backup*, dem Order, in dem sich die Backups aller Systeme befinden. Einen Cronjob habe ich nicht eingerichtet, da *rdfind* erst nach etwa drei Tagen durchgelaufen war. Dieses Verfahren ergibt allem Anschein nach nur Sinn, wenn der Speicherplatz wirklich nicht mehr ausreicht, da es sehr Zeit- und Speicherintensiv ist.

4.2 Entfernung alter Backups

Bei den bisher beschriebenen Aufbauten wird täglich ein Backup erstellt. Nach einiger Zeit wird sich also eine riesige Menge an alten Daten angesammelt haben, die man

höchstwahrscheinlich nie wieder brauchen wird, da sie bereits veraltet und überhaupt nicht mehr relevant sind.

Um diesem Problem entgegenzuwirken, habe ich zwei kleine Script namens *rrb_ cleanup* und *rrb_ cleanup_ helper* geschrieben, die nach konfigurierbaren Regeln eine Liste der nicht mehr benötigten Backups ausgeben. Diese kann man dann nochmal händisch durchgehen um zu überprüfen, ob nicht fälschlicherweise noch benötigte Backups entfernt werden sollen. Oder man gibt die Liste direkt an *rm*[1] weiter, dass die Backups entfernt. Es wären natürlich auch komplexere Anwendungen möglich, wie ein wöchentlicher Bericht darüber, welche Backups gelöscht werden sollen, an den Administrator des Backup-Servers.

Mit den Scripts lassen sich Regeln dafür festlegen, wie viele Backups in welchem Zeitraum behalten werden sollen. Eine solche Regel kann wie folgt aussehen und wird in der Konfigurationsdatei gesetzt:

Listing 4.2: KEEP_RULES für *rrb_ cleanup*

```
1 KEEP_RULES=( \
2     7   7 \
3    31   8 \
4   365  11 \
5 1825   4 \
6 )
```

Die Einträge in den Regeln werden jeweils in Paaren gelesen, deshalb auch die zeilenweise Einteilung. Der erste Wert in jeder Zeile gibt den Zeitraum in zurückliegenden Tagen an, für den die Regel gilt. Der zweite Wert gibt die Anzahl der Backups an, die in diesem Zeitraum behalten werden sollen. Wichtig ist dabei, dass die Regeln der Reihe nach durchgearbeitet werden und bereits durch vorhergehende Regeln behaltene Backups durch spätere nicht mehr berücksichtigt werden. In jedem Zeitraum wird versucht möglichst weit auseinander liegende Backups zu behalten.

Die erste Regel gilt für die letzten sieben Tage und behält sieben Backups, im Idealfall jeweils einen Tag auseinanderliegend. Denn der Quotient aus den Tagen und der Anzahl ist eins. Die nachfolgende Regeln bezieht sich auf einen Monat in dem acht Sicherungen zusätzlich zu den sieben der ersten Regel behalten werden sollen. Dieses Schema lässt sich für die nächsten Einträge für ein Jahr und fünf Jahre fortsetzen.

4.3 Offline-Backup

Obwohl der Backup-Server bereits, wie im Kapitel Sicherheit beschrieben, abgesichert ist, können immer noch durch menschliche, Software- oder Hardwarefehler Sicherungen

[1] Befehlszeile dazu: *rrb_ cleanup /etc/rrb/config.x | xargs rm -rf*

beschädigt werden. Um dem entgegenzuwirken sollten die Backups regelmäßig auf ein Offline-Medium gesichert werden, also ein Medium, das nicht dauerhaft ansprechbar ist.

Hierzu verwende ich eine zweite Festplatte, die bei Bedarf über den eSATA-Anschluss mit meiner Workstation verbunden wird. Über Netzwerk aktualisiere ich dann die Backups auf der externen Festplatte vom Backup-Server. Hierzu ist auf dem Backup-Server ein spezieller Benutzer eingerichtet, der lediglich die Berechtigung hat die Backups auszulesen. Mehr zu diesem Aufbau in der Sektion Benutzer.

Das dabei auf der Workstation verwendete Script ist sehr einfach:

Listing 4.3: Script für Offline-Backup

```
1 #!/bin/sh
2
3 backup_disk_connect
4 rsync -avHP rrb_all@haruka:backup/ /media/backup_disk/backup/
5 backup_disk_disconnect
```

Rsync wird so aufgerufen, dass es die Dateien komplett unverändert überträgt (-a), ausführlich berichtet was es tut (-v), Hardlinks erhält (-H), und den Fortschritt ausgibt (-P).

Alternativ dazu ließe sich auch *xfsdump* verwenden, ein Port des in der Einleitung erwähnten *dump* für das von mir verwendete Dateisystem XFS. Dieses Verfahren wäre vermutlich performanter, da es auf der Dateisystem-Ebene arbeitet. Ich habe es jedoch nicht verwendet, da dafür die Backups auf einer seperaten Partition sein müssten, was sie aber unglücklicherweise nicht sind.

Kapitel 5

Wiederherstellung

Genauso wichtig wie die Sicherung der Daten ist die Möglichkeit der problemlosen Wiederherstellung ebendieser, selbst dann wenn nur minimale Mittel zur Verfügung stehen. Um dies zu erreichen konnten viele Lösungen nicht verwendet werden:

Eine NFS-4 basierte Wiederherstellung könnte im Zweifelsfall fehlschlagen, wenn die nicht gerade einfache Einrichtung der Benutzer-Authentifizierung per Kerberos nicht gelingt, oder Kerberos oder NFS-4 gar nicht auf der Live-CD vorhanden sind. Ein NFS3-Zugriff würde zwar von so gut wie jedem System aus funktionieren, NFS-3 bietet jedoch nur IP-basierte Authentifizierung, und die stellt keine ernsthafte Sicherheit dar.

Ein verschlüsseltes netzwerkbasiertes Dateisystem wie *sfs*[1] ist nur auf wenigen Live-CDs enthalten und auch nur auf einigen Plattformen verfügbar. *Sfs* ist außerdem relativ unerprobt, man kann also nicht unbedingt mit einer zuverlässigen Transferierung der Daten rechnen.

Die einzige noch bleibende Möglichkeit ist eine Wiederherstellung per *ssh*.

5.1 Benutzer

Das Problem dabei ist jedoch, dass die Benutzer dann bei der Wiederherstellung vollständigen Zugriff auf den Server hätten, also fremde Backups auslesen, auf fremde Rechner zugreifen, und den Server manipulieren können. Gibt man den Benutzern keinen root-Zugriff, können sie nicht alle ihrer Dateien wiederherstellen, nämlich die nicht, die nicht von jederman ausgelesen werden können. Außerdem wäre immernoch ein Aufruf von Programmen auf dem Backup-Server und ein Auslesen von anderen lesbaren Daten möglich. Der Benutzer hätte also zu viele Berechtigungen.

Um dem entgegenzuwirken, lässt sich eine beschränkte Shell einrichten. Hierbei bietet sich *scponly* an. Dieses Programm *chroot*et den Benutzer beim Zugriff in sein Home-Verzeichnis. Dadurch kann er auch nur die sich dort befindenden Programme benutzen.

[1] sfs: Self-certifying File System

Und das sind genau die, die notwendig sind, um Daten per *sftp*[2], *scp*[3], *sshfs*[4], und *rsync* heraus zu befördern.

Das nächste Problem ist jedoch, wie der Benutzer, der ja in seinem Home-Verzeichnis eingesperrt ist, auf die Backups zugreifen kann, die ja außerhalb von seinem Home-Verzeichnis liegen und auch nicht vollständig von ihm lesbar sind, da er nicht die Berechtigungen dazu hat. An diesem Punkt habe ich mich dieses Tricks bedient:

Das Backup-Verzeichnis wird auf dem Backup-Server per *nfs* ausschließlich für *localhost*, also den Backup-Server selbst, zur Verfügung gestellt. Dabei werden alle Zugriffe so behandelt, als kämen sie vom root-Account, man kann also ausnahmslos alle Dateien auslesen. Um eine Veränderung der Backups zu verhindern, werden lediglich Leseberechtigungen vergeben. Die Unterordner dieses so zur Verfügung gestellte Backup-Ordners können nun im Home-Verzeichnis des jeweiligen Benutzers ge*mount*et werden.

Diesen gesamten Prozess habe ich automatisiert und das Script rrb_create_account angelegt. Mit diesem Script lässt sich für jeden zu sichernden Rechner ein seperater Benutzer anlegen. Es besteht also auch nicht die Gefahr, dass ein Benutzer die gesicherten Daten eines anderen ausliest.

5.2 Partielle Wiederherstellung

Einen Teil der Daten wiederherzustellen ist sehr einfach und auf mehrere Möglichkeiten machbar. Sofern man direkt weiß welche Daten man benötigt, kann man diese per *scp* vom Server kopieren:

Listing 5.1: Partielles Wiederherstellung mit *scp*

```
1 # scp rrb_x@haruka:backup/2008-12-31T23:59:59/etc/fstab /etc/fstab
2 Password: [Passwort eingeben]
```

Alternativ lässt sich per *sftp* ein Zugriff ähnlich wie mit einem ftp-Client simulieren. Dabei kann man auch in den Backups navigieren und die richtigen Dateien aussuchen:

Listing 5.2: Partielles Wiederherstellung mit *sftp*

```
1 # sftp rrb_x@haruka:backup/
2 Password: [Passwort eingeben]
3 sftp> ls
4 2008-1231T23:59:59  2009-03-01T18:40:03  latest
5 sftp> cd latest/etc
6 sftp> get fstab
```

[2]sftp: secure ftp
[3]scp: secure copy
[4]SSH FileSystem

```
7 Fetching /backup/2009-03-01T18:40:03/etc/fstab to fstab
8 sftp> quit
```

Am komfortabelsten ist der Zugriff per *sshfs*, wie er im folgenden Kapitel beschrieben ist. Er ist natürlich auch zur partiellen Wiederherstellung von Daten geeignet.

5.3 Vollständige Wiederherstellung

Die komplette Wiederherstellung ist nach der in der Sektion Benutzer geleisteten Vorarbeit kein Problem mehr. Angenommen der Rechner x habe einen Festplattenschaden erlitten und müsse neu aufgesetzt werden aus dem letzten erstellten Backup auf dem Backup-Sever *haruka*. Von einer Live-CD sei bereits gebooted und die neuen Dateisysteme bereits erstellt und unter /mnt/new-root/ eingehängt:

Listing 5.3: Vollständige Wiederherstellung von x

```
1 # mkdir /mnt/backup/
2 # sshfs rrb_x@haruka:backup/ /mnt/backup/
3 Password: [Passwort eingeben]
4 # cp -a /mnt/backup/latest/* /mnt/new-root/
5 # dd if=/mnt/new-root/boot/mbr.bck of=/dev/sda bs=512 count=1
```

5.4 Wiederherstellung aus der DMZ

Da, wie bereits erwähnt, aus der DMZ keine Verbindungen in das LAN aufgebaut werden dürfen, funktionieren die soeben vorgestellten Möglichkeiten zur Wiederherstellung von Daten nicht ohne weiteres. Der Backup-Server muss zuerst bestimmte Maßnahmen treffen, bevor ein Zugriff aus einem PC der DMZ auf den Backup-Server möglich ist. Wichtig ist dabei auch, dass aus dem Netz der DMZ lediglich Zugriffe auf die entsprechenden Backups erlaubt sind. Dies lässt sich in der */etc/ssh/sshd_config* einstellen.

Damit der Rechner aus der DMZ auf seine Backups zugreifen kann, muss der Backup-Server einen Port-Forward einrichten, über den der Client dann verbinden kann:

Listing 5.4: Erstellen eines Port-Forward auf dem Backup-Server

```
1 # ssh -R 2222:localhost:22 exxt
```

Von nun an kann *exxt* über den Port 22 auf *localhost* auf seine Backups zugreifen. Ein Zugriff per sftp sieht dann wie folgt aus:

Listing 5.5: Nutzen eines Port-Forwards von *exxt* aus

```
1 # sftp -oPort=2222 rrb_exxt@localhost:backup/
```

Kapitel 6

Schluss

Ich hoffe hiermit meine Backup-Lösung anschaulich erläutert zu haben. Sie wird mittlererweile zur Sicherung von sechs Computern, einschließlich dem Backup-Server selber, verwendet und hat sich soweit bewähren können.[1] Probleme, wie eine immer schlechter werdende Performance oder die Unmöglichkeit auf vernünftige Weise Backups wiederherzustellen, die ich von etablierten Lösungen gewohnt bin, sind bisher nicht aufgetreten.

[1] Für die Firewall wird *rrb* nicht eingesetzt, da sich auf dieser so gut wie nie Daten ändern. Außerdem handlet es sich nur um ein sehr kleines System, bei dem selbst vollständige Backups innerhalb von weniger als einer Minute möglich sind und lediglich 200 MB groß sind.

Kapitel 7

Quelltexte

7.1 rrb

```
1 #!/bin/sh
2
3 # This is the reverse rsync backup (rrb) utility. It has to be run on the
4 # backup server and will then make a backup of a specified directory of a
5 # client machine via ssh, rsh, or any other remote shell or the localhost using
6 # rsync with hardlinks. See the config.example for more information about the
7 # configuration of rrb for a client.
8 #
9 # Further features include an exclusion file, the resume of a failed transfer,
10 # customizable commands to be executed before running, after running, after a
11 # failure, after a success.
12 #
13 # You can use fdupes or a similar tool to hardlink between multiple machines.
14 #
15 # No backups will be done automatically. You have to run rrb by hand every time
16 # you want to make a backup. Use cron to make regular backups, at to make
17 # backups at a specific time, or a custom script using sxc, or portknocking, or
18 # whatever to enable the client to request a backup.
19 #
20 # If you want to automize backups via ssh, you can use public key
21 # authentication. So if you do full-system backups from your backup server, it
22 # will have root access to all the backup clients. Root access on the server
23 # means root access on all the clients and a loss of all the backups. Secure
24 # the server and save backups offline, too.
25 #
26 # The backups can be made accessible in any manner, like NFS, SMB, or a custom
27 # script that tars the files and sends them over the network. Please use your
28 # creativity.
29 #
30 # This script should be compatible to zsh and bash. Tell me if it fails in your
31 # favourite shell.
32
33 # ------------------------------------------------------------------------------
34
35 set -e # Exit on every fail.
36 HELP="Usage: $0 config"
37
38 # Exit codes
39 EX_USAGE=64
40 EX_SOFTWARE=70
41 EX_CONFIG=78
42 EX_FILE=100
43 EX_RUNNING=101
```

```
44
45 # Unset variables from config file.
46 unset SRC EXCLUDES_FILE DEST_DIR BEFORE_CMD AFTER_CMD FAIL_CMD SUCCESS_CMD \
47       RSYNC_OPTS
48
49 fail()
50 {
51   echo -e "$1" >&2
52   exit $2
53 }
54
55 run()
56 {
57   if [ "$*" ]; then
58     echo "$*" | sh
59   fi
60 }
61
62 add_rsync_opt()
63 {
64   if [ "${RSYNC_OPTS}" ]; then
65     RSYNC_OPTS=("${RSYNC_OPTS[@]}" "$*")
66   else
67     RSYNC_OPTS=("$*")
68   fi
69 }
70
71 cleanup()
72 {
73   [ 0 -ne $? ] && run "$FAIL_CMD"
74   run "$AFTER_CMD"
75   rm -f "$LOCK_FILE"
76 }
77
78 [ "$#" -eq 1 ] || fail "Invalid usage.\n$HELP" $EX_USAGE
79 [ "$1" ]       || fail "Config file missing.\n$HELP" $EX_USAGE
80 source $1
81 [ "$SRC" -a "$DEST_DIR" ] || fail "Invalid config file, has to include at \
82 least \$SRC and \$DEST_DIR.\n$HELP" $EX_CONFIG
83
84 cd $DEST_DIR
85
86 LATEST_DIR="latest"
87 TMP_DIR=".tmp"
88 NEW_DIR="`date +%FT%T`"
89 LOCK_FILE=".lock"
90
91 [ -e "$NEW_DIR" ] && fail "Backup directory $NEW_DIR already exists." $EX_FILE
92
93 [ "$EXCLUDES_FILE" ] && add_rsync_opt --exclude-from="$EXCLUDES_FILE"
94 [ -L "$LATEST_DIR" ] && add_rsync_opt --link-dest="$PWD/$LATEST_DIR"
95
96 # noclobber prevents the '>' from overwriting an existing lock file.
97 if ! (set -o noclobber; echo $$ > "$LOCK_FILE"); then
98   fail "$DEST_DIR is already locked by the process with the PID \
99 $(cat "$LOCK_FILE"). Remove $LOCK_FILE to unlock manually." $EX_RUNNING
100 fi
101
102 trap cleanup EXIT HUP INT QUIT TERM # Always call, even on success.
```

```
103 run "$BEFORE_CMD"
104 rsync -avzP --delete --delete-excluded "${RSYNC_OPTS[@]}" "$SRC" "$TMP_DIR" \
105 || [ 24 -eq $? ]
106
107 mv "$TMP_DIR" "$NEW_DIR"
108 rm -f "$LATEST_DIR"
109 ln -sf "$NEW_DIR" "$LATEST_DIR"
110 run "$SUCCESS_CMD"
111 # Call cleanup.
```

7.2 config.example

```
1  # Example configuration file for backing up a client using rrb.
2  # Will be interpreted as a shell script. Comments start with #.
3  # Use KEY=VALUE to assign a value to the key. No spaces around the =.
4  #
5  #-------------------------------------------------------------------------
6  # Obligatory options
7  #-------------------------------------------------------------------------
8
9  # The root of the files to be backuped. Can be on a remote host or localhost.
10 # Will be interpreted by rsync.
11
12 # To backup a local directory (and create an additional directory level at the
13 # destination):
14 #SRC=/etc
15
16 # In order not to create an additional directory level at the destination add a
17 # trailing slash:
18 #SRC=/etc/
19
20 # Remote access via remote shell. Note that you don't need to add a trailing
21 # slash. Spaces in remote sources have to be escaped, so the remote shell can
22 # interpret them. My tip: Don't use spaces at all:
23 #SRC=example:'file\ name\ with\ spaces'
24
25 # Remote access via rsync daemon. No need to add a trailing slash on remote
26 # connections:
27 #SRC=example::module
28
29 # The destination directory to put the backups in:
30 #DEST_DIR=/backup/example
31
32 #-------------------------------------------------------------------------
33 # Optional options
34 #-------------------------------------------------------------------------
35
36 # The path of a file that specifies the excludes. New line for every file to be
37 # excluded. (Directories are files, too.) See man 1 rsync for more detail on
38 # the layout of the file.
39 #EXCLUDES_FILE=/etc/rrb/excludes.example
40
41 # Same with spaces:
42 #EXCLUDES_FILE="/path with spaces/excludes.example"
43
44 # A command that will be executed before running the backup. Can be used to
45 # mount a filesystem.
46 #BEFORE_CMD="echo 'before'"
```

```
47
48 # A command that will be executed after running the backup. Also if the backup
49 # failed. Can be used to mount a filesystem that has been mounted before. Note
50 # that you can't use variables to find out whether the filesystem was mounted
51 # before BEFORE_CMD.
52 #AFTER_CMD="echo 'after'"
53
54 # Will only be executed if the backup failed. Might be handy to notify the
55 # admin.
56 #FAIL_CMD="echo 'fail'"
57
58 # Will only be executed if the backup succeeded.
59 #SUCCESS_CMD="echo 'success'"
60
61 # Set additional rsync options. In this example we want to preserves hard
62 # links. Preserving hard links is very expensive, and we don't want to hinder
63 # the users of the client too much, so we set the niceness of the remote rsync
64 # to a high value.
65 #RSYNC_OPTS=( --hard-links --rsync-path='nice -n19 rsync' )
66
67 # Specify a remote shell for rsync. There are more rsync-related options, see
68 # man 1 rsync.
69 #RSYNC_RSH=ssh
70
71 # ------------------------------------------------------------------------------
72 # Options for rrb-cleanup
73 # ------------------------------------------------------------------------------
74
75 # Number of days to keep the backups and number of backups to keep in this
76 # interval. One rule per line. Backups recognized by earlier lines are excluded
77 # from following lines. Therefore you usually want to write the number of days
78 # in ascending order.
79 #KEEP_RULES=( \
80 #    7   7 \
81 #   31   8 \
82 #  365  11 \
83 #1825   4 \
84 #)
```

7.3 RRB_INTERVAL

```
 1 #!/bin/sh
 2
 3 # Run rrb if it was last run more than the specified number of seconds ago.
 4 # Useful in cron-scripts for clients, that are not online all the time.
 5
 6 set -e
 7 HELP="Usage: $0 interval config"
 8 EX_USAGE=64
 9 EX_CONFIG=78
10 unset DEST_DIR
11
12 fail()
13 {
14   echo -e "$1" >&2
15   exit $2
16 }
17 do_rrb()
```

```
18 {
19     exec rrb $CONFIG
20 }
21
22 INTERVAL="$1"
23 CONFIG="$2"
24 [ "$#" -eq 2 ]         || fail "Invalid usage.\n$HELP" $EX_USAGE
25 [ "$INTERVAL" ]        || fail "Interval missing.\n$HELP" $EX_USAGE
26 [ "$INTERVAL" -ge 1 ] || fail "Invalid interval.\n$HELP" $EX_USAGE
27 [ "$CONFIG" ]          || fail "Config file missing.\n$HELP" $EX_USAGE
28 source $CONFIG
29 [ "$DEST_DIR" ]        || fail "Invalid config file, missing \$DEST_DIR.
30 $HELP" $EX_CONFIG
31
32 LATEST_DIR="$DEST_DIR/latest"
33 [ -e "$LATEST_DIR" ] || do_rrb
34 # date does not parse datetime with 'T' as delimiter correctly, replace with
35 # space. (Or it does it correctly by recognizing the timezone, but this is not
36 # wanted here)
37 DATE_LAST=`date --date="\`readlink $LATEST_DIR | tr T ' '\`" +%s`
38 DATE_LIMIT=`echo "\`date +%s\` - $INTERVAL" | bc`
39 [ $DATE_LAST -le $DATE_LIMIT ] && do_rrb
```

7.4 RRB_CLEANUP

```
1 #!/bin/sh
2
3 # Cleanup a user's backup directory by printing no longer needed backups
4 # (seperated by null charaters) using a defined policy. Pipe to xargs -0 rm -rf
5 # to really delete them.
6
7 set -e
8 HELP="Usage: $0 config"
9 EX_USAGE=64
10 EX_CONFIG=78
11 unset DEST_DIR KEEP_RULES
12
13 fail()
14 {
15   echo -e "$1" >&2
16   exit $2
17 }
18
19 CONFIG="$1"
20 [ "$#" -eq 1 ]     || fail "Invalid usage.\n$HELP" $EX_USAGE
21 [ "$CONFIG" ]      || fail "Config file missing.\n$HELP" $EX_USAGE
22 source $CONFIG
23 [ "$DEST_DIR" ]    || fail "Invalid config file, missing \$DEST_DIR.
24 $HELP" $EX_CONFIG
25 [ "$KEEP_RULES" ] || fail "Invalid config file, missing \$KEEP_RULES.
26 $HELP" $EX_CONFIG
27
28 exec rrb_cleanup_helper "$DEST_DIR" "${KEEP_RULES[@]}"
```

7.5 RRB_CLEANUP_HELPER

```
1 #!/usr/bin/env python
```

```
2
3 import sys
4 import os
5 import os.path
6 import datetime
7
8 def execRule(now, rule, backups):
9   oldestDt = now - datetime.timedelta(days=rule[0])
10  vBackups = []
11  for backup in backups:
12    if backup[0] > oldestDt:
13      vBackups.append(backup)
14  if len(vBackups) < rule[1]:
15    del backups[:]
16    print >> sys.stderr, "Rule", rule, "keeps all"
17    return
18
19  optDts = [now - datetime.timedelta(days=y) * rule[0] / rule[1] \
20    for y in sorted(range(rule[1]))]
21  for optDt in optDts:
22    diffs = []
23    for backup in vBackups:
24      diffs.append((abs(optDt - backup[0]), backup[0], backup[1]))
25    diffs.sort()
26    print >> sys.stderr, "Rule", rule, "tries to get", \
27      optDt.strftime("%Y-%m-%dT%H:%M:%S"), "and keeps", diffs[0][2]
28    backups.remove((diffs[0][1], diffs[0][2]))
29    vBackups.remove((diffs[0][1], diffs[0][2]))
30
31 def main(argv=None):
32  if not argv:
33    argv = sys.argv
34
35  if len(argv) < 5 or len(argv) % 2 != 0:
36    print >> sys.stderr, "Usage: " + argv[0] + " dir rule1_days rule1_nr [...]"
37
38  baseDir = argv[1]
39  rules = map(lambda x: (int(x[0]), int(x[1])), zip(*[iter(argv[2:])]*2))
40
41  dirs = os.listdir(baseDir)
42  backups = []
43
44  for dir in dirs:
45    try:
46      dt = datetime.datetime.strptime(dir, "%Y-%m-%dT%H:%M:%S")
47    except ValueError:
48      continue
49    backups.append((dt, dir))
50  backups.sort(reverse=True)
51
52  now = datetime.datetime.now()
53
54  for rule in rules:
55    execRule(now, rule, backups)
56
57  # Print backups to be deleted.
58  for backup in backups:
59    sys.stdout.write(os.path.join(baseDir, backup[1]) + '\0')
60
```

```
61 if __name__ == '__main__':
62   main()
```

7.6 RRB_CREATE_ACCOUNT

```
1  #!/bin/sh
2
3  SETUP_CHROOT_PATH=/etc/scponly/chroot/
4
5  set -e
6  HELP="Usage: $0 name backup-dir"
7
8  EX_USAGE=64
9
10 fail()
11 {
12   echo -e "$1" >&2
13   exit $2
14 }
15
16 [ "$#" -eq 2 ] || fail "Invalid usage.\n$HELP" $EX_USAGE
17 [ "$1" ]       || fail "Username missing.\n$HELP" $EX_USAGE
18 [ "$2" ]       || fail "Backup directory missing.\n$HELP" $EX_USAGE
19 NAME="$1"
20 BACKUP_DIR="$2"
21 cd "$SETUP_CHROOT_PATH"
22 echo "$NAME" | ./setup_chroot.sh
23 HOME_DIR="`grep "^$NAME" /etc/passwd | cut -d ":" -f6`"
24 passwd "$NAME"
25 rmdir "$HOME_DIR/incoming"
26 mkdir "$HOME_DIR/backup" "$HOME_DIR/dev"
27 mknod -m 666 "$HOME_DIR/dev/null" c 1 3
28 echo "localhost:$BACKUP_DIR $HOME_DIR/backup nfs defaults 0 0" >> /etc/fstab
29 mount "$HOME_DIR/backup"
```

7.7 SXC-RRB

```
1  #!/bin/sh
2
3  # Backups using sxc and rrb.
4
5  HELP="Usage: $0 config"
6
7  # Exit codes
8  EX_USAGE=64
9
10 fail()
11 {
12   echo -e "$1" >&2
13   exit $2
14 }
15
16 [ "$#" -eq 1 ] || fail "Invalid usage.\n$HELP" $EX_USAGE
17 [ "$1" ]       || fail "Config file missing.\n$HELP" $EX_USAGE
18 source $1
19
20 main()
```

```
21 {
22   PID=$1
23
24   while [ ! -p $JID/in ]; do
25     echo "Waiting for $JID/in"
26     sleep 0.5
27   done
28
29   echo -e "\0:pwd $PASSWORD\0" > $JID/in
30   echo -e "\0:pri $PRIORITY\0" > $JID/in
31   echo -e "\0:set available\0" > $JID/in
32
33   while [ "`cat $JID/nfo/presence`" = "offline" ]; do
34     sleep 1
35   done
36
37   LAST_USER=
38   LAST_CONFIGS=()
39   for ENTRY in ${USERS[@]}; do
40     if echo "$ENTRY" | grep -qs "@"; then
41       if [ "$LAST_USER" ]; then
42         echo -e "\0:add $LAST_USER\0" > $JID/in
43         handle_contact $PID "$LAST_USER" "${LAST_CONFIGS[@]}" &
44       fi
45       LAST_USER="$ENTRY"
46       LAST_CONFIGS=()
47     else
48       if [ "${LAST_CONFIGS}" ]; then
49         LAST_CONFIGS=("${LAST_CONFIGS[@]}" "$ENTRY")
50       else
51         LAST_CONFIGS=("$ENTRY")
52       fi
53     fi
54   done
55
56   echo -e "\0:add $LAST_USER\0" > $JID/in
57   handle_contact $PID "$LAST_USER" "${LAST_CONFIGS[@]}" &
58
59   tail --lines=0 --follow --pid $PID $JID/out 2> /dev/null |
60   while read -r line; do
61     OUT=`echo $line | cut -d" " -f 2-` # Remove date
62     if echo $OUT | grep -qs "^Disconnected"; then
63       ( sleep 5; echo -e "\0:set available\0" > $JID/in) &
64     fi
65   done
66 }
67
68 handle_contact()
69 {
70   PID=$1
71   CONTACT=$2
72   CONFIGS=${*:3}
73   FILE_OUT=$JID/$CONTACT/out
74   FILE_IN=$JID/$CONTACT/in
75
76   while [ ! -p $FILE_IN -a -f $FILE_OUT ]; do
77     echo "Waiting for $CONTACT/{in,out}"
78     sleep 0.5
79   done
```

```
80
81    tail --lines=0 --follow --pid $PID $FILE_OUT 2> /dev/null |
82    while read -r line; do
83      OUT=`echo $line | cut -d" " -f 2-` # Remove date
84
85      # We are only interested in messages from the other contact.
86      if ! echo $OUT | grep -qs "^<$CONTACT> "; then
87        continue
88      fi
89
90      MSG=`echo $OUT | cut -d" " -f 2-`
91
92      CMD=`echo $MSG | cut -d" " -f 1`
93      if [ ":help" = "$CMD" ]; then
94        echo "\
95 Commands:
96    :backup name
97    :list" > $FILE_IN
98      elif [ ":list" = "$CMD" ]; then
99        echo "Valid hosts: ${CONFIGS[@]}" > $FILE_IN
100     elif [ ":backup" = "$CMD" ]; then
101       NAME=`echo $MSG | cut -d" " -f 2-`
102
103       for CONFIG in ${CONFIGS[@]}; do
104         if [ "$CONFIG" = "$NAME" ]; then
105           VALID=true
106         fi
107       done
108       if [ "$VALID" = "true" ]; then
109         echo "Running: rrb \"$CONFIG_PREFIX$NAME$CONFIG_SUFFIX\"" > $FILE_IN
110         sleep 0.1
111         rrb "$CONFIG_PREFIX$NAME$CONFIG_SUFFIX" &> $FILE_IN
112         echo "rrb \"$CONFIG_PREFIX$NAME$CONFIG_SUFFIX\" finished with exit code: $?" \
113         > $FILE_IN
114       else
115         echo "Invalid hostname $NAME" > $FILE_IN
116       fi
117     else
118       echo "Invalid usage, see :help" > $FILE_IN
119     fi
120   done
121 }
122
123 sxc $JID --port $PORT &
124 PID=$!
125 main $PID &
126 wait $PID
```

7.8 SXC-RRB-SETTINGS

```
1 JID=rrb@example.org
2 PASSWORD=insecure
3 PRIORITY=0
4 PORT=5222
5
6 # Prefix for every config file.
7 CONFIG_PREFIX=/etc/rrb/config.
8 # Suffix for every config file.
```

```
 9 CONFIG_SUFFIX=
10
11 # Entries including an "@" are the JIDs, all following entries the config files
12 # that user is allowed to request a backup for.
13 USERS=( \
14    foo@example.org comp1 \
15    bar@example.org comp2 comp1 \
16    xyz@example.org comp2 comp3 \
17 )
```