

# SelfLinux-0.13.0



## Domain Name System



Autor: Steffen Dettmer ([steffen@dett.de](mailto:steffen@dett.de))  
Formatierung: Peter Schneewind ([peter@schneewind.net](mailto:peter@schneewind.net))  
Lizenz: GFDL

Dieses Dokument soll den Einstieg in DNS erleichtern. Es wendet sich an zukünftige DNS-Administratoren von kleinen Netzwerken.

Kommentare, Korrekturen und Hinweise sind willkommen! Dazu bitte eine kurze eMail an: *Steffen Dettmer* <[steffen@dett.de](mailto:steffen@dett.de)> schicken.

## Inhaltsverzeichnis

**1 Einführung**

**2 Resolver**

**3 Bind**

**4 Zonen**

**5 DNS-Datenbank und deren Einträge**

**6 Meister und Sklaven**

**7 Resource Records**

**8 Start of Authority - Ein Beispiel**

**9 Die benötigten Resource Records für unsere Zone**

**10 Die andere Seite - Reverse Lookups**

**11 Eine Reverse Lookup Zone**

**12 Zonefiles**

**13 Abkürzungen**

**14 Die fertigen Dateien**

**15 Konfiguration des Bind**

**16 ROOT-SERVERS konfigurieren**

**17 Eigene Zones konfigurieren**

**18 Eine Zone für "localhost"**

**19 Weitere wichtige Konfigurationsoptionen**

**20 Starten**

**21 Zoneänderungen und Secondaries**

**22 Testen der Konfigurationen**

**23 Spielzonen**

**24 Mehr Informationen als Namen und Adressen**

**25 Quickstart**

**26 Ein Produktivsystem**

**27 Sinnvolle SOA-Values**

**28 Secondaries betreiben**

**29 Zugriffsbeschränkungen konfigurieren**

**30 Betrieb hinter einer Firewall**

**31 Classless Routing (CIDR)**

**32 Umgang mit Störungen**

**33 Root-Rechte**

**34 Chroot-Umgebung**

**35 Versionen**

**36 Wachstum**

**37 Keep it running**

## 1 Einführung

Da sich ein Mensch IP-Nummern schlecht merken kann, benutzt man lieber Namen. Nun braucht man einen Dienst, der diese irgendwie in IP-Adressen übersetzen kann.

Zunächst führte man (auf jedem Host) die Datei `/etc/hosts` ein, in der, durch Whitespace getrennt, eine IP-Adresse, ein Name und beliebig viele Aliasnamen stehen. Der Name ist der "erste, wirkliche, echte Systemname", die Aliasnamen sind zusätzliche. In der Praxis kann das natürlich auch umgekehrt sein, denn manche Administratoren verwechseln die Namen und geben dem System mit dem Kommando `hostname` einen Namen, der eigentlich "nur" ein Alias ist. Dies kann unter Umständen zu Problemen führen.

Eine `/etc/hosts` kann zum Beispiel so aussehen:

/etc/hosts	
127.0.0.1	localhost
192.168.1.1	ns1.selflinux.de
192.168.1.2	ns2.selflinux.de
192.168.1.3	rebecca.selflinux.de www.selflinux.de

Neben dem `localhost` Eintrag, der stets eingetragen sein sollte, sind hier noch zwei Namensserver und ein Webserver eingetragen. Der Webserver heißt eigentlich `rebecca`, `www` ist ein Zweit- oder Aliasname.

Damit Namen auf allen Maschinen bekannt sind, müssen die Einträge in allen `/etc/hosts` auf allen Maschinen stehen. In den Anfängen des ARPA-Net wurde diese Datei als `hosts.txt` dann auf alle Maschinen verteilt. Diese wurde schnell groß und unübersichtlich. Um Konflikte zu vermeiden, kam man auf die Idee, jeder Organisation einen Namen zu geben, der durch einen Punkt dahinter geschrieben wurde. Damit blieben die Namen eindeutig, auch wenn zwei Organisationen einen Host Namens `FTP` haben wollten, denn hinter dem Punkt waren sie unterschiedlich.

Später kam noch eine weitere Unterteilung hinzu, nämlich die nach Ländern. So gibt es jetzt einen Server `www` in der Organisation `selflinux` in Deutschland, kurz `www.selflinux.de`. Die Organisation bezeichnet man allgemeiner als **Domain**, was im mathematischen Sinne das "Gemeinsame" bedeutet.

Da das für tausende von Maschinen nicht mehr wartbar ist, erfand man eine verteilte Datenbank, die das verwaltet, und nannte es das **Domain Name System**, kurz **DNS**. Verteilte Datenbank bedeutet, dass jeder nur einen Teil verwaltet. Eine einfache und hier günstige Struktur ist die Baumstruktur, die angewendet wird.

An der Wurzel stehen die sogenannten **Top Level Domains**, **TLDs**, z.B. `de`, `com`, `org`. Diese werden von Wurzelservern bedient, also dem Teil der Datenbank, der darüber Informationen hat. Englisch heißen diese **ROOT-SERVERS**.

Darunter kommen dann die Server, welche die TLD-Datenbanken haben, es sind mehrere, um ausfallsicher zu sein und die Last zu verteilen. Diese haben dann Informationen über die Organisations-Domains, die man zusammen mit der TLD kurz mit **Domain** bezeichnet, eine Domain wäre z.B. `selflinux.de`.

Eine Domain hat in der Regel einen Namensserver, der nun endlich die Namen wie `www` kennt. Das nennt man **Delegation**.

Da DNS-Server aus Performancegründen Namen cachen (also zwischenspeichern), kann es auch sein, dass ein anderer Namensserver `www.selflinux.de` kennt, aber er hat die Information "aus zweiter Hand". Die Server, die die Daten wirklich haben, meist eine Datei auf einer Platte, und deren Antwort verbindlich ist, nennt man **autoritativ**. Bekommt man von einem DNS-Server eine Antwort aus dem Cache, so ist sie als **non-autorativ** gekennzeichnet, d.h. sie ist vielleicht nicht mehr gültig.

Ein TLD Server ist für eine TLD (z.B. `de`) autoritativ.

Die "Rootdomain", die Vater/Mutterdomain, nennt man ".".

Ein ROOT-SERVER ist also für "." autoritativ.

Die ROOT-SERVER delegieren nun die Autorität zu den TLD-Servern. Der ROOT-SERVER "weiß" (und bestimmt), welcher DNS-Server für z.B. `de` autoritativ ist. Ein `de`-Server enthält nun auch nicht die Daten über `www` der Domain `selflinux`, sondern er delegiert die Autorität zu einem anderen DNS-Server, der nun unter `selflinux.de`; eintragen kann, was er will, z.B. `www`.

## 2 Resolver

Ein Programm, z.B. ein WWW-Browser, muß nun natürlich so einen DNS-Server fragen können. Damit das nicht jedes Programm von sich aus implementieren muß, ist das Bestandteil der libc. Dies nennt man **Resolver**, zu deutsch: "Auflöser". Die Funktionen verwenden Konfigurationsdateien (bzw. bei Win entsprechende Registryeinträge), um z.B. die IP Adresse des zu verwenden Namensservers zu ermitteln. Der Resolver sendet eine Anfrage an einen DNS-Server, und erwartet die (endgültige) Antwort. Der DNS-Server muß nun hinten beginnend den Namen auflösen. Soll er z.B. [www.selinux.de](http://www.selinux.de) für den Klienten auflösen, muß er sich dabei rekursiv durch die DNS-Struktur hangeln.

Er fragt zuerst einen ROOT-SERVER nach den DNS-Servern (falls noch nicht im Cache), die für [de.](http://de.) autoritativ sind. Dann fragt er einen dieser nach den für [selinux.de.](http://selinux.de.) autoritativen, und einen von diesen nach [www.selinux.de.](http://www.selinux.de.) Hat er die Antwort im Cache, oder hatte ein anderer DNS-Server (der von "unserem" als **Forwarder** benutzt wird, der also die Anfragen weitergereicht bekommt) die Antwort im Cache, so wird sie dem Client als **non-authoritative** gekennzeichnet übermittelt. Diese Information bekommt das Programm normalerweise allerdings nicht zu sehen.

Es gibt auch DNS-Server, die diese rekursiven Resolveranfragen nicht auflösen. Diese können von einem Client nicht als DNS-Server benutzt werden. Das reduziert die Belastung dieser DNS-Server, was z.B. bei den ROOT-Servern wichtig ist.

Häufig wird unter UN\*X der BSD-Resolver verwendet. Dieser wird mit der Datei `/etc/resolv.conf` konfiguriert. Hier können einige Optionen gesetzt werden. Eine ist natürlich die zu verwendenden Namensserver. Diese werden einfach untereinander (jeder in einer Zeile) aufgezählt (z.B.: "nameserver 127.0.0.1" usw.). Zusätzlich wird hier die DNS-Domain eingetragen (z.B.: "domain self-linux.de"), und eine Suchliste (z.B.: "search selinux.de de com" - ein benutzerfreundliches Beispiel).

Es wird immer der erste Namensserver gefragt, außer wenn dieser ausfällt. Nach einem großzügigen Timeout wird dann der zweite verwendet.

Es gibt eine Erweiterung, der **Splitted Resolver**. Hier kann man zusätzlich angeben, für welche Domains der Namensserver verwendet wird (die Option "nameserver" erhält dies als weiteren Parameter). Diese Erweiterung ist allerdings relativ selten anzutreffen. Die Suchliste funktioniert wie folgt:

Wird ein Name nicht gefunden, so wird der erste Teil der Suchliste angehängt (also [selinux.de](http://selinux.de)). Wird auch der so erweiterte Name nicht gefunden, so wird dem ursprünglichen der zweite Teil angehängt, und eine dritte Anfrage gestartet usw. Der Client (also das Programm) bemerkt davon nichts. So kann man sich etwas Schreibarbeit sparen. Eine Beispieldatei wäre also:

<code>/etc/resolv.conf</code>
<pre>domain selinux.de search selinux.de de com nameserver 127.0.0.1 nameserver 192.168.1.53</pre>

Sicherheitshalber sollte man stets mehrere Nameserver angeben.

Verwendet man Windows-Clients, so sind diese Parameter im entsprechenden Fenster einzustellen. Dabei ist zu beachten, dass Windows Namen grundsätzlich zuerst über das Windows-Namensprotokoll aufzulösen versucht, was zu unerwarteten Effekten führen kann.

In einer weiteren Datei (`/etc/nsswitch.conf`) legt man unter moderneren UN\*X-Varianten fest, in welcher Reihenfolge welche Methoden zur Namensauflösung verwendet werden (lokale Dateien, DNS oder NIS). Die entsprechende Option heißt "hosts". In einer typischen DNS-Konfiguration (ohne NIS) ist sie meist auf

<code>/etc/nsswitch.conf</code>
<code>hosts: files dns</code>

gesetzt (erst lokale Datei: `/etc/hosts`, dann DNS verwenden).

## 3 Bind

Im folgenden wird `bind` in einer 8er-Version vorausgesetzt. `bind` ist der Nameserver-Deamon, der vom Internet Software Consortium gepflegt und entwickelt wird. Es sind zwar noch hier und da Versionen 4.x im Einsatz, diese sterben jedoch aus. Neue Server sollten immer eine aktuelle Version erhalten, und auch bestehende sollten hin und wieder geupdatet werden, insbesondere, wenn Sicherheitslücken bekannt geworden sind!

## 4 Zonen

Nun gibt es einen weiteren wichtigen Begriff: **Zone**. Ein Zone wird gerne mit einer Domain verwechselt, da muß man aufpassen, die Unterschiede sind wichtig.

In Kurzform: Eine Zone ist das, für das ein DNS-Server autoritativ ist, also etwas **Delegiertes**.

Ein DNS-Server ist immer für eine oder mehrere Zones maßgebend (nämlich die, die zu ihm delegiert wurden). Umgekehrt gibt es zu jeder Zone einen DNS-Server, der für diese autoritativ ist. Nun gibt es aber sehr große Zones, z.B. die der Universität Berkeley `berkeley.edu`. Die DNS-Admins von Berkeley delegieren also weiter.

Zum Beispiel könnte man die großen Zweige `cs` und `math` delegieren. Die bekommen einen eigenen Server und der Admin von `cs` muß nicht immer die Admins von Berkeley auffordern, einen Namen einzutragen, das kann er selbst machen. Damit sind `cs` und `math` Zonen der Domain Berkeley. Diese bezeichnet man auch als **Subdomains**, da sie unter Berkeley liegen.

Wir haben nun eine Domain, die aus mehreren Zonen besteht, und jede Zone gehört zu genau einer Domain (nämlich Berkeley). Berkeley selbst ist auch eine Zone, aber diese enthält nur wenige Informationen, nämlich die Adressen der DNS-Server, die für `cs` und `math` maßgebend sind, und vielleicht noch ein paar Namen aus kleinen Subdomains wie z.B. `art`. Diese Subdomain liegt nun - im Gegensatz zu `cs` - in der Zone Berkeley! Das heißt also, `cs` hat einen eigenen Namensserver, ist also eine Zone, `art` jedoch nicht. Hier liegt also der Unterschied, denn beides sind Subdomains, aber nur `cs` ist eine Subzone.

## 5 DNS-Datenbank und deren Einträge

Eine DNS-Datenbank besteht aus zwei wichtigen Typen von Einträgen: Zuordnung von Namen zu Adressen (Addressrecords "A"), und Einträgen, die einer Adresse einen Namen zuordnen, also Zeigern auf einen Namen (Reverserecords "PTR"). Daneben gibt es aber noch weitere. Diese Einträge liegen in einer Zone, die von einem Namensserver autoritativ verwaltet werden. Diese Einträge selbst nennt man **Resource Record**, kurz **RR**.

Neben Namen und IP-Adressen gibt es einen RR (also "Typ") für die Eigenschaften einer Zone. Es gibt RRs, die Textinformationen beinhalten, die geographische Position festlegen und weitere.

Die DNS-Datenbank ist also eine verteilte Datenbank, die in Zonen unterteilt ist. Diese Zonen werden von DNS-Servern bedient. Eine Zone besteht aus Einträgen, die man Resource Records nennt.

## 6 Meister und Sklaven

Da man für große Zonen mehrere DNS-Server benötigt, können diese Datenbanken übertragen werden. Auf einer Maschine werden die Dateien gepflegt, und dann auf die anderen kopiert. Diese eine Maschine nennt man "Master" oder **Primary**, die anderen "Slave" oder **Secondary**. Die letzteren sind die moderneren Formen. "Slave" ist irreführend, denn ein Slave kann für einen anderen "Slave" auch "Master" sein, und außerdem ist ein Slave auch autoritativ für diese Zone.

Die DNS-Server von großen Zonen kann man in der Regel gar nicht fragen, da sie stark abgesichert sind, dazu sind die Secondaries da.

Von "außen" kann man nicht erkennen, ob man einen Master oder einen Slave fragt, beide sind genauso autoritativ für die Zone. Ein Slave legt die Zonendaten auch auf seiner Platte ab, damit er auch nach einem Neustart Antworten liefern kann, wenn der Master nicht erreichbar ist. Hier liegt der große Unterschied zum Caching. Daten aus dem Cache werden beim Neustart verworfen.

Daten aus dem Cache können niemals autoritativ sein.

Der DNS-Server bestimmt auch einige Eigenschaften der Zone selbst, z.B. wie lange die Zone gecached werden darf, und in welchen Zeitabständen die Secondaries prüfen sollen, ob sich die Zone geändert hat. Diese Daten liegen in einem ganz speziellen Record, dem **Start of Authority**, kurz **SOA**. Dieser legt den Beginn und die Eigenschaften einer Zone fest.

## 7 Resource Records

Ein Record besteht aus mehreren Teilen: dem **Objekt** (Datenbankleute nennen es auch das Schlüsselfeld), der **TTL** (Time to live, die maximale Zeit, die es gecached werden darf), der **Klasse**, zu der der Record gehört, der **Typ** des Records (z.B. "A" für eine Namen-Address-Zuordnung) und dem **Inhalt**.

Die TTL gibt an, wie lange ein Objekt gecached werden darf, also wie lange es in einem Cache lebt. Üblich sind Werte von drei oder besser acht Stunden.

Die Klasse gibt an, zu welcher Netzwerkart der Name gehört. Das Internet heißt **IN**, das ist in 99,99% der Fälle die verwendete Klasse, die wenigen Ausnahmen (Chaosnet und Hesiod - damit man es mal gehört hat, früher gab es auch mal CSNET) werden hier nicht behandelt, bis auf eine kleine Ausnahme.

Der Typ wurde schon erwähnt: Es gibt Namen, Adressen, Texte (die wichtige Zusatzinformationen enthalten können), Aliasnamen, Hostinformationen, Mailserver und andere. Einige davon gibt es praktisch fast nur "auf dem Papier".

## 8 Start of Authority - Ein Beispiel

Die Zone `selflinux.de`. (ab jetzt schreiben wir immer, wenn wir sicher sind, den vollen Namen zu meinen, einen "." hinten dran, den Namen der Root-Zone) könnte so aussehen (nur der SOA RR):

```
selflinux.de. IN SOA [Inhalt]
```

Das Objekt ist die Zone. Es gehört zur Klasse Internet, und es handelt sich um eine Zone, genauer gesagt, um den Start einer Autorität.

Nun müssen hier etliche Informationen drinstehen:

- \* Der Name des primären Nameservers (Ursprungsserver)
- \* Eine EMailadresse, falls jemand mal eine Frage oder einen Hinweis hat. Hier wird anstatt des "@" Zeichens ein "." verwendet (ja, hier darf eine EMailadresse im ersten Teil keinen Punkt enthalten!)

Dann in Klammern:

- \* Eine Seriennummer der Zonenversion, an der die Secondaries erkennen, ob sich was geändert hat
- \* Informationen, wann die Secondaries diese Zonendaten aktualisieren sollen (refresh)
- \* Wie lange sie warten sollen, wenn das nicht klappt (retry)
- \* Nach welcher Zeit die Daten definitiv ungültig sind (expire)
- \* Wie lange ein RR mindestens gültig ist (min. TTL).

Die Zeiten werden alle in Sekunden angegeben. Da sich die Seriennummer stets erhöhen muß, hat es sich bewährt, das aktuelle Datum im Format JJJJMMTT und eine zweistellige tägliche laufende Nummer zu verwenden. So erkennt man auch gleich, wann die Zone zum letzten Mal verändert wurde.

Damit sieht ein SOA so aus (";" trennt Kommentare):

## named.hosts

```
selflinux.de. IN SOA pri-ns.selflinux.de. admin.selflinux.de. (  
    2000012501 ; Serial (Seriennummer)  
    10800      ; Refresh (Aktualisierung) 3 Stunden  
    3600       ; Retry (neuer Versuch)  
    864000    ; Expire (ungültig nach 10 Tagen)  
    86400 )   ; min. TTL (Mindestgültigkeit) 1 Tag
```

DNS ist nicht case-sensitiv, `.de.`, `.De.`, `.DE.` ist also das gleiche. Deshalb schreibe ich RR-Typen und Klassen groß, Namen klein, um besser unterscheiden zu können.

## 9 Die benötigten Resource Records für unsere Zone

In dieser Zone gibt es selbstverständlich DNS-Server, die diese Angaben weitergeben können. Diese RR's heißen **NS** - Name Server, das sieht dann so aus:

selflinux.zone
<pre>selflinux.de. IN NS ns1.selflinux.de. selflinux.de. IN NS ns2.selflinux.de.</pre>

Wir könnten auch noch pri-ns dazuschreiben, aber wir möchten nicht, dass dieser viel gefragt wird, also lassen wir ihn weg.

Nun finden sich in der `/etc/hosts` (beispielsweise) folgende Einträge:

/etc/hosts
<pre>192.168.1.1 ns1.selflinux.de 192.168.1.2 ns2.selflinux.de 192.168.1.3 rebecca.selflinux.de www.selflinux.de</pre>

Der "localhost" Eintrag paßt nicht in die Zone "selflinux.de", da "localhost" ja nicht mit "selflinux.de" endet.

rebecca hat einen Zweitnamen "www". Zweitnamen sind vom RR Type **CNAME** (canonical name). Dabei gilt es zu beachten, dass CNAMEs nie "auf der rechten Seite stehen dürfen". Das bedeutet, es gibt keine CNAMEs von CNAMEs, MX- oder NS-Records "zeigen" nicht auf CNAMEs, sondern immer auf A-Address-Records.

Diese nun übersetzt in RRs:

selflinux.zone
<pre>ns1.selflinux.de.      IN A 192.168.1.1 ns2.selflinux.de.      IN A 192.168.1.2 rebecca.selflinux.de.  IN A 192.168.1.3 www.selflinux.de.      IN CNAME rebecca.selflinux.de.</pre>

Hier sind keine TTLs (Cache-Zeiten) angegeben, es wird die min TTL aus dem SOA Record verwendet. Wir könnten das aber machen, z.B.:

selflinux.zone
<pre>www.selflinux.de.      3600 IN CNAME rebecca.selflinux.de.</pre>

Der WWW-Name darf nur eine Stunde gecached werden (vielleicht steht die Umstellung auf eine andere Maschine bevor?). Das senkt die Performance, führt zu höherem Traffic, darf also nicht versehentlich drin stehen

bleiben.

Für die gesamte Zone sei ein Mailserver verantwortlich, z.B. rebecca. Mit dem **MX**-RR definiert man einen Mail Exchanger und dessen Priorität. Niedrigere Prioritäten werden dabei vorgezogen. Ein Eintrag könnte demnach lauten:

selflinux.zone	
selflinux.de.	IN MX 10 rebecca.selflinux.de.

Damit bekommt rebecca Mail für <user>@selflinux.de. Natürlich muß das dortige Mailprogramm eMail dieses Aussehens annehmen, und nicht weitersenden (z.B. Klasse "Cw" bei `sendmail` muß diese Namen "selflinux.de" [als Hostnamen] beinhalten). Zusätzlich kann für jeden Host mittels Wildcard "\*" dieser Mailexchanger erzwungen werden:

selflinux.zone	
*.selflinux.de.	IN MX 10 rebecca.selflinux.de.

Dabei ist zu beachten, dass das auch für garnicht existierende Hosts matched, man kann also EMail an sowas wie <user>@halleluja.ax123.selflinux.de senden.

## 10 Die andere Seite - Reverse Lookups

Nun kann hosts aber mehr, und zwar auch rückwärts: IP in Namen "auflösen". Das nennt man **reverse-lookups**. Um nicht ein neues Schema (neben Domains und Zones) zu benötigen, hat man einfach die Zones benutzt, das Konzept ist ja flexibel - schließlich kann man nicht alle Domains durchfragen, bis man die Adresse gefunden hat!

Diese Art der Auflösung wird vor allem bei sicherheitsempfindlichen Diensten verwendet. Das Tool `nslookup` (siehe auch Abschnitt "Testen der Konfigurationen") beispielsweise prüft, ob die IP-Adresse zu einem DNS-Server existiert (macht also ein Reverse Lookup), und auch NFS-Server lösen Client-IP-Adressen rückwärts auf. Der Sinn darin ist, DNS Spoof-Angriffe zu erschweren.

Bei diesen Angriffen werden DNS-Namen gefälscht, beispielsweise durch einen Eindringling auf einem DNS-Server. Da häufig für Vorwärts- und Rückwärtsauflösung verschiedene Server und vor allem verschiedene Delegationen verwendet werden, muß ein Angreifer häufig die Kontrolle über zwei DNS-Server erlangen, hat also etwas schlechtere Chancen.

Dafür gibt es einen neuen Domain-Namensraum, der nicht Namen als Domains benutzt, sondern IP-Adressen. Dieser ist ziemlich unabhängig vom "normalen" Namensraum! Diese Domain hat man "arpa" genannt (eben aus historischen Gründen). Da es für alle Klassen (z.B. Internet) eigene Adressen gibt, ist auch diese Teil des Namens. Internet-Adressen sind **in-addr**, also heißt die Domain "in-addr.arpa.". Dieser Baum enthält dann letztlich alle IP-Adressen. Diese muß man natürlich delegieren können!

Dabei gibt es eine Besonderheit: die allgemeineren, die Netzwerknamen (Domainnamen) stehen ja hinten, z.B. ist "www" ein Host im Netzwerk "selflinux.de.". Bei IP-Adressen steht das Netzwerk aber vorn, der Hostanteil hinten! Bei der Delegation wird ein bestimmter Teil (etwas "Allgemeineres", z.B. eine Domain) delegiert. Wenn man das jetzt mit IP-Adressen macht, kann man nicht den ersten Teil einer IP-Adresse delegieren, denn dann hätte man ein "Loch" - den Hostanteil - das erst in einer tieferen Instanz gefüllt werden könnte! Also muß man die IP-Adresse byteweise umdrehen, so wird aus "192.168.1.2" "2.1.168.192". Nun steht der Netzwerkanteil hinten, und man kann einen Teil delegieren, wie auch bei Domainnamen.

Das kann man wie bei Namen an jedem "." machen. Die Zone "in-addr.arpa." enthält also 255 Zonen, eine davon ist "192.in-addr.arpa.", diese enthält wiederum 255 Zonen usw. Rebecca liegt also in Zone "1.168.192.in-addr.arpa.". Diese benötigt erst einmal einen SOA Record. Namensserver und EMail seien wie oben.

## 11 Eine Reverse Lookup Zone

1.168.192.zone
<pre>1.168.192.in-addr.arpa. IN SOA pri-ns.selflinux.de. admin.selflinux.de. (     2000012501 ; Serial (Seriennummer)     10800      ; Refresh (Aktualisierung) 3 Stunden     3600       ; Retry (neuer Versuch)     864000    ; Expire (ungültig nach 10 Tagen)     86400 )   ; min. TTL (Mindestgültigkeit) 1 Tag</pre>

Die Nameserver schreibt man auch einfach dazu:

1.168.192.zone
<pre>1.168.192.in-addr.arpa. IN NS ns1.selflinux.de. 1.168.192.in-addr.arpa. IN NS ns2.selflinux.de.</pre>

Nun die drei eigentlichen Einträge. Man hat jetzt **PTR**-Records, weil wir ja rückwärts auflösen:

1.168.192.zone
<pre>1.1.168.192.in-addr.arpa.      IN PTR ns1.selflinux.de. 2.1.168.192.in-addr.arpa.      IN PTR ns2.selflinux.de. 3.1.168.192.in-addr.arpa.      IN PTR rebecca.selflinux.de.</pre>

(wieder ohne TTLs)

## 12 Zonefiles

Da das später Zonefiles für `bind` werden sollen, müssen diese in Dateien geschrieben werden. Standardmäßig liegen diese in `/var/named/` - auch die von Primären Servern. Über die Namen der Dateien gehen die Meinungen auseinander, insbesondere, wenn es um Reverse-Lookup-Dateien geht. Viele sind gewohnt, bei Dateien einen Postfix anzuhängen (".txt", ".c"), deshalb kann man hier ".zone" verwenden.

Hier wird der Domainname + ".zone" verwendet, also "selflinux.zone" und "1.168.192.zone" (wenn man nur in-addr für Reverse braucht, also vermutlich alle bis auf ein paar Jungs vom MIT).

Bei Secondary Servern (Slaves) werden diese Dateien nicht per Hand angelegt. Sie werden vom Primary geholt, wenn es nötig ist, also wenn die Refresh-Zeit um ist, oder der Primary eine entsprechende Nachricht schickt. Die Daten werden dann in die Dateien geschrieben, damit sie notfalls auch einen Neustart "überleben".

Für jede Zone gibt es eine Datei, und in jeder Datei darf nur ein SOA vorkommen. Wir brauchen also bereits zwei, und da wir auch vermutlich "localhost" auflösen können möchten, wie im Prinzip fast jeder DNS-Server, sind wir schon bei vier.

## 13 Abkürzungen

Unsere Zonefiles sind nun ziemlich unübersichtlich, aber bei `bind` sind einige Abkürzungen möglich. `bind` weiß ja (wir werden es ihm später sagen), welche Domain in welcher Datei liegt.

bei dem Eintrag:

selflinux.zone	
<code>ns1.selflinux.de.</code>	<code>IN A 192.168.1.1</code>

Beispielsweise ist ja klar, dass er zu `".selflinux.de."` gehört (sonst wäre es die falsche Datei!). Das kann man weglassen. Auch die Klasse kann man weglassen, dann wird `"IN"` angenommen, man erhält:

selflinux.zone	
<code>ns1</code>	<code>A 192.168.1.1</code>

Die Domain wird an alle Namen angehängt, nicht jedoch an IP-Adressen, die hinter einem `"A"` stehen. Aber an Namen, die z.B. hinter einem `"CNAME"` stehen, es sei denn, diese sind explizit als vollständig angegeben und enden mit einem `"."`.

Beim SOA Record ist das ähnlich:

selflinux.zone	
<code>selflinux.de.</code>	<code>IN SOA pri-ns.selflinux.de. admin.selflinux.de. (...)</code>

Für die Domain `"selflinux.de"`, in deren Datei wir uns ja befinden, kann man auch `"@"` schreiben, also:

selflinux.zone	
<code>@</code>	<code>IN SOA pri-ns.selflinux.de. admin.selflinux.de. (...)</code>

Die Domain wird auch hier angehängt, deshalb geht auch:

selflinux.zone	
<code>@</code>	<code>SOA pri-ns admin (...)</code>

aber es sieht nicht mehr sehr gut aus... Der Autor mag z.B. gar keine Abkürzungen in SOA Records, aber das ist Geschmackssache.

Es gibt noch eine weitere Abkürzung: Gibt es mehrere RRs für dasselbe Objekt, so muß dieses nur einmal (beim ersten RR) genannt werden. Die weiteren werden dann so angenommen, also ist auch erlaubt:

selflinux.zone	
selflinux.de.	IN NS ns1.selflinux.de.
	IN NS ns2.selflinux.de.

Da hier diese Einträge direkt nach dem SOA Record kommen (der ja ein RR für "selflinux.de." ist), kann man bei 1. NS RR (also beim 2. RR der Zone) dieses auch noch weglassen, was in der Praxis meistens gemacht wird.

Auch muß man (für `bind`, andere DNS-Server machen das vielleicht anders!) die Zeiten im SOA und TTLs nicht unbedingt in Sekunden angeben, man schreibt statt "3600" einfach "3H". Es gibt für Tage (days) "D", für Wochen (weeks) "W" und für Minuten (minutes) "M".

Verwendet man keine TTL-Angaben bei den Records, wird ein Defaultwert verwendet. Ältere `bind`-Versionen verwenden hierzu stets min. TTL aus dem SOA Record, neuere kennen dazu die Direktive `$TTL`, die im Zonefile eingestellt wird. In den Beispielen wird hier immer eine Default-TTL von einem Tag verwendet: `$TTL 1d`.

## 14 Die fertigen Dateien

Nun seien die folgenden beiden Dateien in `/var/named`:

```
/var/named/selflinux.zone

$TTL 1d
selflinux.de.  IN SOA pri-ns.selflinux.de. admin.selflinux.de. (
    2000012501 ; Serial (Seriennummer)
    3H        ; Refresh (Aktualisierung)
    1H        ; Retry (neuer Versuch)
    1M        ; Expire (ungültig nach)
    1D )      ; min. TTL (mindeste Gültigkeit)

                IN NS    ns1.selflinux.de.
                IN NS    ns2.selflinux.de.

ns1             IN A     192.168.1.1
ns2             IN A     192.168.1.2
rebecca        IN A     192.168.1.3
www            IN CNAME rebecca.selflinux.de.
```

```
/var/named/1.168.192.zone

$TTL 1d
1.168.192.in-addr.arpa. IN SOA pri-ns.selflinux.de. admin.selflinux.de. (
    2000012501 ; Serial (Seriennummer)
    3H        ; Refresh (Aktualisierung)
    1H        ; Retry (neuer Versuch)
    1M        ; Expire (ungültig nach)
    1D )      ; min. TTL (mindeste Gültigkeit)

                IN NS    ns1.selflinux.de.
                IN NS    ns2.selflinux.de.

1              IN PTR    ns1.selflinux.de.
2              IN PTR    ns2.selflinux.de.
3              IN PTR    rebecca.selflinux.de.
```

(ein hohes Expire ist in der Praxis durchaus sinnvoll)

Diese sollen nun von `bind` als Primäre Zonen verwaltet werden, er soll autorativ sein, dafür sind die Dateien entwickelt worden.

## 15 Konfiguration des Bind

Nun müssen wir `bind` entsprechend konfigurieren. Wie schon im ersten Teil wird dabei auf `bind`-Version 8 und höher eingegangen.

Bei Neuinstallationen sollte keinesfalls mehr `bind 4` verwendet werden!

Beginnen wir mit einer typischen Konfiguration für ein kleines Netzwerk. Wir stellen nur einen DNS-Server auf, also soll dieser alle DNS-Funktionen erfüllen: Er soll unsere Zonen halten, von Clients/Resolvern als DNS-Server verwendet werden können, und einen Cache aufbauen können, um Datenverkehr zu sparen.

Die Syntax der Konfigurationsdatei ist ähnlich zu C++. Die Datei besteht aus Anweisungen, die jeweils mit einem Semikolon abgeschlossen werden. Anweisungen können auch Blöcke sein. Es gibt Blöcke, die einen Namen haben. Der Block selbst wird in `{ }` geklammert.

Eine wichtige Anweisung bzw. ein wichtiger Block ist `options {}`; Hier stehen die globalen Optionen. Der andere wichtige Blockname ist `zone`. Zwischen Zone und den `{ }` Klammern steht der Name der Zone, also `'zone "selflinux.de" {}`;'.

Kommentare können wie in C++ in `"/*` und `*/` gefaßt werden, oder nach `"/` bis zum Zeilenende reichen.

Wichtige Anweisungen im Optionsblock sind:

```
                                /etc/named.conf

directory "/var/named" ;      // Verzeichnis mit den Zonefiles
forwarders {1.2.3.4;         // Welche DNS Server als
    1.2.3.5;};               // Forwarder verwenden?
// forward only;            // nicht selbst auflösen,
                             // sondern den Forwarder das
                             // machen lassen
forward first;               // den Forwarder fragen, wenn
                             // der es nicht weiß, selbst
                             // rekursiv nachfragen
```

Den Rest lassen wir erstmal weg, verwenden also die Defaults. Diese wurden beim Compilieren festgelegt. Wenn man ein Paket seiner Distribution verwendet, sollten diese gut zu den anderen passen.

Es sei an dieser Stelle ausdrücklich darauf hingewiesen, dass man keine Forwarders verwenden muß.

## 16 ROOT-SERVERS konfigurieren

Wenn der DNS-Server selbst rekursiv fragen können soll (also in allen Fällen, außer "forward only;"), muß er natürlich die ROOT-SERVER kennen. Das sind die Server für die Zone ".". Diese Zone konfiguriert man in der Regel wie folgt:

```
                                /etc/named.conf

zone "." {
    type hint;                // ist keine "richtig" Zone
    file "root.hint";        // Datei mit den ROOT-SERVERS
};
```

In "root.hint" stehen die "A" (Address) Records (RRs) der ROOT-SERVER, das sieht z.B. so aus:

```
                                /var/named/root.hint

.                3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000  A   198.41.0.4

.                3600000  NS  B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000  A   128.9.0.107
```

Das sollte jetzt verständlich sein: Für "." gibt es in der Klasse IN einen Nameserver mit sehr langer TTL, der A.ROOT-SERVERS.NET. heißt, und dessen IP-Adresse.

Da sich die IP-Adressen auch ändern können, muß diese Datei gepflegt werden. Dazu eignet sich das Dienstprogramm `dig` aus der BIND-Distribution.

Hat man bereits ein System, das funktionierend Namen auflösen kann (z.B. wenn es einfach einen DNS-Server vom Provider in der `/etc/resolv.conf` konfiguriert hat, die "dns" in `/etc/nsswitch.conf` hinter "hosts" steht), braucht man nur ein:

```
user@linux / $ dig . ns
```

zu machen, und erhält eine aktuelle Liste. Man kann auch explizit einen DNS-Server angeben, mit:

```
user@linux / $ dig @ns.isp.com . ns
```

Breibt man einen nur cachenden Server (mindestens ein solcher gehört eigentlich in jedes Netzwerk, um die Performance zu erhöhen), ist damit die Konfiguration im Prinzip fertig. Hat man gute Verbindung zu DNS-Servern mit gutem Cache (also DNS-Server, die häufig gefragt werden), so kann man diese als Forwarder konfigurieren. Hat man die besondere Situation eines sich über Wählleitungen einwählenden Netzes oder auch Hosts, macht es Sinn, die DNS-Server des Providers als Forwarders zu verwenden (wie man es auch als Client tun würde - außer natürlich, diese sind überlastet). Forwarders sind auch nützlich, wenn der DNS-Server öfter heruntergefahren wird. Verwendet man mehrere verschiedene Provider, so verwendet man am besten keine Forwarder, um Probleme mit Zugriffsbeschränkungen zu vermeiden.

## 17 Eigene Zones konfigurieren

Die Zones, für die der Server authoritative ist (also wenn er Primary oder Secondary ist, das hieß früher irreführend Master bzw. Slave), müssen jetzt in der Konfigurationsdatei aufgeführt werden:

```
/etc/named.conf  
  
zone "selflinux.de" {  
    type master;  
    file "selflinux.zone"; // wir sind Primary  
};  
  
zone "1.168.192.in-addr.arpa" {  
    type master;  
    file "1.168.192.zone";  
};
```

Bei Secondaries konfiguriert man "type slave;" und ebenfalls ein File (z.B. file "slave/myzone.zone");. Zusätzlich muß man dann aber noch den (die) Masterserver angeben: "masters {1.2.3.4;}";.

## 18 Eine Zone für "localhost"

Der Name "localhost" sollte immer mit der IP 127.0.0.1 verknüpft sein. Um dies per DNS zu erreichen, gibt es zwei grundlegende Herangehensweisen, die in der Praxis auftauchen. Einmal denkt man sich "localhost" zu einer Zone zugehörig, zum Beispiel als "localhost.selinux.de.". In diesem Fall wird der Name einfach als Address-Record in das entsprechende Zonefile eingetragen.

Eine andere Möglichkeit ist, den Namen als "localhost." zu interpretieren. Aus Sicht des DNS handelt es sich dabei um eine Toplevelzone (mit einer IP-Adresse, aber keinen weiteren darunterliegenden Records). Das verletzt allerdings etwas das Konzept von den Toplevel-Domains, funktioniert in der Praxis jedoch sehr gut. Um dies zu erreichen, erzeugt man ein entsprechendes Zonefile:

```
                                /var/named/localhost.zone

$TTL 1d
localhost. IN SOA pri-ns.selinux.de. admin.selinux.de. (
    2000012501 ; Serial (Seriennummer)
    3H        ; Refresh (Aktualisierung)
    1H        ; Retry (neuer Versuch)
    1M        ; Expire (ungültig nach)
    1D )      ; min. TTL (mindeste Gültigkeit)

                IN NS    ns1.selinux.de.
                IN NS    ns2.selinux.de.

                IN A     127.0.0.1
```

Eine zugehörige Reverse-Zone:

```
                                /var/named/0.0.127.zone

$TTL 1d
0.0.127.in-addr.arpa. IN SOA pri-ns.selinux.de. admin.selinux.de. (
    2000012501 ; Serial (Seriennummer)
    3H        ; Refresh (Aktualisierung)
    1H        ; Retry (neuer Versuch)
    1M        ; Expire (ungültig nach)
    1D )      ; min. TTL (Mindestgültigkeit)

                IN NS    ns1.selinux.de.
                IN NS    ns2.selinux.de.

1                IN PTR  localhost.
```

In der named.conf werden beide Zonen eingetragen:

```
/etc/named.conf

zone "localhost" {
    type master;
    file "localhost.zone";
};

zone "0.0.127.in-addr.arpa" {
    type master;
    file "0.0.127.zone";
};
```

Wird die erste Methode verwendet, also localhost als Host einer Domain interpretiert, so wird nur die Reverse-Zone benötigt (da es ja keine Toplevel-Zone "localhost." gibt). In dieser muß der PTR-Record dann konsistenterweise auf die Domain zeigen, beispielsweise "localhost.selflinux.de.". Man kann auch eine Mischung aus beiden Methoden konfigurieren. Dazu wird eine Zone "localhost." erzeugt, und ein Hosteintrag in der Domain. Dieser Eintrag im Zonefile der Domain ist sinnvollerweise ein CNAME auf "localhost.":

```
/var/named/localhost.zone

localhost      IN CNAME localhost.
```

Die Reverse-Zone sieht damit wie im Beispiel aus.

Vergleicht man beide Methoden, stellt man fest, dass deren Funktion von der Konfiguration der Clients abhängt. Normalerweise muß eine Maschine in der Lage sein, "localhost" aufzulösen. Verwendet man die erste Methode, muß also eine Suchliste definiert sein, bei der unter mindestens einer Domain der Host "localhost" existiert. Hier liegt also ein Vorteil der zweiten Methode.

Der Hauptnachteil der zweiten Methode ist, dass man eine "localhost."-Topleveldomain definiert, was das Toplevel-Konzept etwas verletzt, und dadurch von einigen Administratoren als unsauber bezeichnet wird. In der Praxis hat sich jedoch die zweite Methode bewährt, die zudem etwas performanter ist, und findet breiten Einsatz.

## 19 Weitere wichtige Konfigurationsoptionen

Hier ganz kurz noch ein paar wichtige Optionen:

named.conf	
<pre>listen-on port 53 { 1.2.3.4; }; //Port 53 des angebenen                                 //Interfaces verwenden. Sinnvoll,                                 //wenn man z.B. Dialup-Interfaces                                 //hat, die nicht von Clients query-source address * port 53; //Port 53 beim _Senden_ verwenden                                 //sinnvoll hinter Firewalls.                                 //Zusätzlich kann auch eine                                 //Interfaceadresse angeben                                 //werden, sinnvoll bei Multihomed-Hosts notify yes;                      //bei Änderungen an den                                 //Datenbanken dies den recursion no;                    //Secondaries mitteilen                                 //Verwendet man am besten                                 //bei                                 //Servern, die nur Primary sind                                 //Clients können                                 //diesen Namen nicht verwenden! check-names master warn;        //sind Zone-RRs falsch, nur warnen,                                 //nicht abbrechen check-names master fail;        //Abbruch, wenn Zone fehlerhaft ist allow-query { any; };           //Jeder darf fragen allow-update { none; };         //keiner darf updaten (ändern) allow-update { 1.2.3.4; };      //1.2.3.4 darf updaten allow-transfer { 1.2.3/24; };   //nur Netz 1.2.3.0/24 darf                                 //Zonetransfers machen</pre>	

Die ersten vier Optionen dürfen nur im "options {};"-Block auftreten, die anderen hier genannten Optionen können auch in "zone {};"-Blöcken auftauchen, so dass sich beispielsweise Zugriffsrechte pro Zone einstellen lassen. "notify" sollte bei Slaves auf "no" stehen, da diese sonst dem Primary Zonenänderungen überflüssigerweise mitteilen.

## 20 Starten

`bind` wird normalerweise über ein kleines rc-script gestartet. Man kann `bind` starten, indem man `"/pfad/named"` aufruft, z.B. `/usr/sbin/named`. Beenden geht unter Linux einfach mit `killall named`. Aber dabei ist zu beachten, dass andere Systeme wie z.B. Solaris ein anderes `killall` haben - dieses killt wirklich alles, hier müßte man mit `ps -ef|grep named` die PID des Prozesses ermitteln, und diesen mit `kill <pid>` beenden.

Eleganter geht das mit dem Dienstprogramm `ndc`: zum Starten verwendet man `ndc start`, zum Beenden `ndc stop`.

Man muß nun sorgfältig die Logfiles des `syslogd` prüfen, um Fehler zu erkennen. Die Fehlermeldungen sind in der Regel sehr aussagekräftig, und führen schnell zur Ursache. Schwierig zu erkennen ist aber z.B. eine vergessene geschweifte Klammer in der `named.conf`, besonders für ungeübte und nicht-C++-Kenner. Wird eine Zone angemockert, so muß das Zonefile untersucht werden.

Sehr ungünstige Werte innerhalb des SOA-Records führen z.B. zu einer Warnung. In diesem Fall sagt `bind` aber sogar, was ihn stört, so dass man den Fehler schnell korrigieren kann. Problematischer sind Fehler in der Art von "Cannot find NS on...". Das deutet darauf hin, dass der NS für eine Zone nicht erreichbar ist, oder nicht existiert. Es muß hier der Name eines DNS-Servers stehen, der autoritativ für die Zone ist! Dieser Fehler wird häufig nicht sofort erkannt.

Wird ein Server von einem anderen als autoritativ für diese Zone genannt, und ist dieser genannte Server aber selbst nicht der Meinung, autoritativ für diese Zone zu sein, nennt man ihn **lame server**. Wären alle "lame", verfällt die Zone mit Ablauf der Cache- bzw. Expirezeiten, man merkt es also nicht unbedingt sofort, aber es ist ein schweres Problem.

Hat man die Zonefiles geändert, und die Seriennummer erhöht, müssen die Datenbanken neu geladen werden, dazu macht man ein:

```
user@linux / $ ndc reload
```

und prüft natürlich wieder die Logfiles.

## 21 Zoneänderungen und Secondaries

Ist "notify yes;" konfiguriert, so sendet `BIND` bei jedem Reload eine Nachricht an die Secondaries, dass die Zone modifiziert wurde. Die Secondaries holen daraufhin den SOA-Record, und schauen, ob sich die Seriennummer erhöht hat. Wenn ja, so wird die Zone vom Primary transferiert, ansonsten passiert nichts. Deshalb darf man nie vergessen, die Seriennummer zu erhöhen!

Der Vorgang, der von den Secondaries beim Empfang einer Notify Message stattfindet, ist der selbe wie der, der beim Erreichen der Refresh-Time einer Zone stattfindet.

## 22 Testen der Konfigurationen


Es gibt ein Tool, mit dem man fast beliebige DNS-Anfragen machen kann, um z.B. Konfigurationen zu testen. Dieses ist Teil der `bind`-Distribution und heißt `nslookup`. Dieses versteht viele Optionen als Parameter, kann aber auch interaktiv verwendet werden.

Um die Wirkung der Searchlist zu unterdrücken, sollten wiederum vollqualifizierte Hostnamen mit einem anschließenden Punkt verwendet werden. Mit diesem Tool fragt man nach Daten der Zone, z.B. "www.selflinux.de." Folgende Optionen stehen im interaktiven Modus zur Verfügung (hier teilweise abgekürzt):

- \* "server <server>", z.B. `server ns.selflinux.de`  
Die Anfragen werden an <server> gesendet.
- \* "set querytype=<RR Typ>", z.B. `set q=any`  
Anfragen nach RR-Typ machen
- \* "set debug" / "set nodebug"  
Es werden viele zusätzliche Informationen angezeigt, sehr hilfreich bei der Fehlersuche
- \* "set recurse" / "set norecurse"  
Rekursive oder nicht rekursive Anfragen machen
- \* "ls <zone>", z.B. `ls selflinux.de`  
Zonentransfer machen, und diese auflisten  
`ls` kennt mehrere Optionen, z.B. "-t <RR Type>"

Beim Testen muß natürlich auch nach IP-Adressen gefragt werden, um zu testen, ob diese richtig zurück in Hostnamen übersetzt werden.

Ein sehr gutes automatisches Tool für derartige Tests ist `dnswalk`, insbesondere bei großen Zones ist es sehr nützlich. Dieses erfordert `Perl5`, und das Modul `Net::DNS`. Dieses kann, falls erforderlich, mit der CPAN-Shell einfach und meist unproblematisch installiert werden (`perl -MCPAN -e shell`, nach der Konfiguration genügt in der Regel ein `install Net::DNS`).

`dnswalk` findet man unter:  <http://www.cis.ohio-state.edu/~barr/dnswalk/> Beim Aufruf übergibt man mindestens die zu testende Zone (mit abschließendem Punkt). `DNswalk` kann auch rekursiv delegierte Zones durchlaufen.

Das Tool `nslookup` hat aber auch einige Nachteile, die das Debugging erschweren können. Beispielsweise versucht `nslookup` stets, zumindest in neueren Versionen, den Namen des DNS-Servers rückwärts aufzulösen. Dies ist oft sinnvoll, um DNS-Spoof-Angriffe zu erschweren. Bei Aufsetzen eines DNS-Systems funktioniert damit das Tool jedoch erst, wenn die Reverse-Zonen konfiguriert sind, und die DNS-IPs richtig auflösen. Um "nur" einfache Tests der Namen zu machen, kann man jedoch andere Tools verwenden, zum Beispiel die, die den normalen Resolver verwenden (beispielsweise zeigt auch `ping` die IP-Adresse an).

`bind` kann auch sämtliche Anfragen via `syslog(3)` loggen. Dazu verwendet man das Dienstprogramm `ndc` (das Kontroll-Interface), um es einzuschalten:

```
user@linux / $ ndc querylog
```

Nun kann man verfolgen, welche Namen aufgelöst werden sollen, welche Server dazu verwendet werden, usw. Das ist auch bei Dial-On-Demand-Verbindungen hilfreich. Ein Blick in die Manpage von `ndc` verrät außerdem weitere Kommandos. Eines davon ist `dumpdb`:

```
user@linux / $ ndc dumpdb
```

Das sorgt dafür, dass die gesamte Datenbank in eine Datei geschrieben wird. Deren Name kann mit der "dump-file"-Direktive im "options {}"-Block gesetzt werden, ansonsten wird `/var/tmp/named_dump.db` verwendet. Diese Datei enthält alle autoritativen Zonendaten und den gesamten Cache, d.h. sie ist ziemlich groß (ein normaler Server produziert dabei einige tausend Zeilen!). Dort kann man z.B. die Zonen überprüfen, oder nach evtl. falsch gecachten Einträge suchen (wenn z.B. zu klären ist, welcher Server eine bestimmte Anfrage falsch beantwortet), in der Praxis wird das aber vermutlich die Ausnahme sein.

## 23 Spielzonen

Nach *RFC 2606* darf man ".test", ".example", ".localhost" und ".invalid" als TLD verwenden. Diese TLDs sind für entsprechenden Gebrauch reserviert, wobei ".example" für Beispiele und Dokumentationen und ".test" für Software-Tests empfohlen ist. ".localhost" ist für 127.0.0.1 reserviert und sollte nicht anders verwendet werden.

## 24 Mehr Informationen als Namen und Adressen

Es gibt RR-Typen, die weniger technischen Aufgaben dienen, sondern allgemeine Informationen liefern. Zunächst gibt es den Typ **HINFO**, das steht für "Host Information". Damit besteht die Möglichkeit, einen Maschinennamen (Hardware-Namen) und einen Betriebssystemnamen aufzunehmen. Historisch gesehen sollte es dazu dienen, um Dienste von den richtigen Maschinen zu verlangen (z.B. HTTP vom WWW-Server), deshalb sollten diese Werte Abkürzungen aus *RFC 1700* (oder Nachfolger) sein, oder auch eigene Abkürzungen. In der Praxis findet dieser RR-Typ nicht die breite Verwendung.

Die Verwendung dieses Records veröffentlicht Informationen, die die Sicherheit beeinflussen können, da ein potentieller Attacker mehr Informationen besitzt, die er gegen das Netz einsetzen kann.

Ein Beispiel wäre:

selflinux.zone	
sun12	IN HINFO "Sun Enterprise 450" "SunOS 5.6"

Für die Dienstzuordnung dient auch der **SRV**-Record, der in *RFC 2052* beschrieben wird. Dabei wird eine zu einem Namen **Assigned Number** (d.h. laut *RFC 1700*) verwendet, die als Prefix vor den Namen gesetzt wird. Sucht ein Client z.B. einen FTP-TCP/IP-Dienst in der Domain selflinux.dx, so würde er nach ftp.tcp.selflinux.de fragen. Ein solcher Datensatz enthält dann die folgenden Felder: Priorität (niedrige werden vorgezogen), Wichtigkeit (größere Server bekommen höhere Nummern), Portnummer (auf der Zielmaschine), Zielmaschine. Ein Beispiel wäre:

selflinux.zone	
nntp.tcp	IN SRV 10 0 119 sun12

Dieser RR findet anscheinend in der Praxis kaum Verwendung.

Eine weitere Art der Information ist der geographische Standort. Dieser kann mit einem **LOC**-RR dargestellt werden. Dieser enthält dabei geographische Breite, Länge und die Höhe über dem Meeresspiegel. Das Format des Datensatzes ist Grad, Minuten, Sekunden, N/S (Nord/Süd), Grad, Minuten, Sekunden (Komma erlaubt, z.B. "4.12"), W/E (West/East), Höhe (in Metern). Sekunden und Minuten dürfen weggelassen werden.

Bei der Ermittlung der Werte hilft  <http://www.ckdhr.com/dns-loc/> und sehr gut ist auch  <http://www.mapblast.com/>. Ein Beispiel (Der Ort sei Lat.: 52 56' (etwa 52,9) N, Long.: 12 49' (12,8) E):

selflinux.zone	
selflinux.de	LOC 52 55 N 12 50 E 100m

Dahinter kann dann noch die Größe, die vertikale, die horizontale Genauigkeit kommen (alle drei in Metern). Damit kann man z.B. die Genauigkeit senken, in dem man sie nur grob auf ein paar tausend Meter angibt, um Sicherheitsrisiken zu vermeiden. Näheres siehe *RFC 1876*.

Diese Informationen können dann z.B. von graphischen `traceroute`-ähnlichen Programmen verwendet werden, die dann auf einer (Welt-)Karte die Routenwege einzeichnen.

## 25 Quickstart

Es gibt noch einen ganz anderen, "unorthodoxen" Weg zu einem DNS-Server. Man schreibt einfach alle Daten in die `/etc/hosts`, und verwendet das Tool `h2n`. Dieses erzeugt aus der hosts Zonefiles und sogar ein Konfigurationsfile für `bind`. Es ist durchaus keine schlechte Idee, dieses Tool zum ersten Erstellen der Files zu verwenden, und diese dann weiterzupflegen. Der Autor hat es jedoch nie benutzt, so dass an dieser Stelle nicht näher darauf eingegangen werden kann.

## 26 Ein Produktivsystem

Bis jetzt war alles Spaß. Für einen Produktionsserver sind noch einige Dinge zu beachten. Im folgenden ein paar Tips, die das Leben hoffentlich vereinfachen.

Im Allgemeinen muß man natürlich vor allem sorgfältig arbeiten. Die meisten Leute sind es sowieso gewohnt, dass DNS-Einträge bis zu drei Tage dauern. Man sollte sich also Zeit lassen, nicht "eben mal auf die Schnelle ein paar RRs einfügen". Man sollte immer Backups von den Datenbanken haben. Dabei bietet sich ein Revision Control System an, z.B. `rcs` oder `cvcs`. Verwaltet man mehrere Zonen, so kann man darüber nachdenken, ein Script zu erstellen, das `dnswalk` für alle Zonen automatisch machen kann. Evtl. kann man den Primary direkt nach ausgesuchteten, sehr wichtigen Adressen per `cron` fragen, um im Fehlerfalle schnell eine EMail zu bekommen.

Es ist auch zu beachten, dass sich unter Umständen Fehler erst nach einigen Tagen zeigen, da die Cachezeiten u.U. relativ groß sind.

Es ist sinnvoll, die Option "check-names fail;" beim Primary zu setzen, um Fehler schnellstmöglich festzustellen, bevor ebtl. falsche Daten von Secondaries geholt werden. Im Extremfalle verwenden diese diese Option, und werfen die fehlerhafte Zone nach dem Transfer sofort weg!

Um sicher zugehen, nach jeder Änderung die Serial richtig zu setzen, kann ein kleines Script verwendet werden. Dies erscheint sinnvoll, da die Serial eine wichtige Rolle spielt. Da man in der Regel das Datum im Format JJJMMTT plus eine zweistellige laufende Nummer verwendet (falls man mehrmals am Tag etwas ändert), wird diese zwangsläufig immer größer. Ein großes Problem aber ist z.B., wenn man sich vertippt, und eine Ziffer zuviel verwendet. Dann hat man eine Serial, die so groß ist, dass sie nie "richtig" erreicht wird. Eine Serial zurückzusetzen, ist so einfach jedoch nicht möglich.

Ein derartiges Script könnte z.B. nach der Zeichenkette "; Serial" suchen, die nach einem entsprechenden Datumscode steht (dieser Kommentar wurde auch hier in den Beispielen verwendet). Verwendet man dazu `perl`, könnte man zu folgender Lösung kommen:

## new\_serial.pl

```
#!/usr/bin/perl -pi.ser
#Y2K stable - Perl RULES :) <steffen@dett.de>

#Starten mit einem Bind Zone File als Paramter. Es wird ein
#Backup erzeugt (Endung .ser), und alle Zeilen der Form
#"JJJJMMTTNN ; Serial", z.B. "2000012501 ; Serial" angepaßt
#aufs aktuelle Datum bzw. die laufende Nummer erhöht.

#start with bind zone files, it will replace serial of the form
#"YYYYMMDD00 ; Serial" i.e. "2000012501 ; Serial"
#to new date or incrementing last two digits

my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime(time);
my $date = sprintf("%4d%02d%02d", 1900+$year, $mon+1, $mday);

if (m/((20|19)\d{6})(\d{2})(\s+\.:\s+[Ss]erial\s+)/) {
    my $old_date = $1;
    my $old_serial = $3;
    my $rest = $4;
    my $serial = $old_serial;
    if ($old_date eq $date) {
        $serial++;
    } else {
        $serial = 0;
    }
    $serial = sprintf("%02d", $serial);
    s/${old_date}${old_serial}${rest}/${date}${serial}${rest}/;
    print STDERR "Serial found! Old: [$old_date $old_serial] --> ",
        "Changed to [$date $serial]\n";
}
```

Es mag Sinn machen, das so aktualisierte File von diesem Script auch gleich ins RCS/CVS einzuchecken bzw. zu "committen" (`ci -l <file> / cvs commit <file>` - letzteres geht in der Regel allerdings nicht als "root").


## 27 Sinnvolle SOA-Values

Viele machen sich kaum Gedanken um die Wahl günstiger SOA-Werte. Um Ressourcen zu sparen, sollten diese nicht zu klein sein. Hohe TTLs haben dafür den Nachteil, dass im Falle einer Umstellung viel Zeit vergeht, bis die neuen Namen verteilt sind. Da man oft schon im voraus weiß, wann eine solche Umstellung beginnt, kann man z.B. drei Tage vorher die TTL für die betroffenen RRs senken, z.B. von acht auf zwei Stunden. Einen Tag nach der Umstellung erhöht man sie einfach wieder (Erhöhung der Seriennummer nicht vergessen).

Bei sehr statischen Zonen kann man das Refresh auch relativ hoch ansetzen, und auch, wenn das Notify von allen Secondaries unterstützt und in der Zone eingesetzt wird. Hier sind dann durchaus Werte von 24 Stunden sinnvoll. Kann man Notify nicht zu allen Secondaries einsetzen, muß man allerdings für einen zügigen Betrieb diesen Wert senken, z.B. auf drei Stunden.

Das Expire sollte immer hoch angesetzt werden. Kann ein Secondary seinen Primary nicht erreichen, wird nach diesem Zeitraum die Zone ungültig, d.h. sie wird nicht mehr beantwortet. Bei einem geringen Expire, z.B. nur zwei Tage, könnte ein Ausfall eines einzigen Servers, nämlich des Secondaries, über das Wochenende das DNS-System lahmlegen, ein sehr unangenehmer Gedanke. Deshalb sollte man als absolutes Minimum eine Woche setzen!

## 28 Secondaries betreiben

Stellt man DNS für eine Zone, so sind die Vorschriften des Betreibers der übergeordneten Zone zu beachten. Betreibt man z.B. eine "de."-Domain, wie "selflinux.de.", so gelten die Vorschriften des  [DENIC](#). Eine sehr sinnvolle ist z.B., dass eine Zone über mindestens zwei, über verschiedene Wege zu erreichende authoritative Server verfügen muß.

Normalerweise hat man einerseits den Primary und andererseits braucht man mindestens einen Secondary in einem anderen Netzwerk. Für einen guten ISP sollte es kein Problem darstellen, einen Secondary zu betreiben. Es scheint auch eine gute Idee zu sein, zwei Secondaries vom ISP machen zu lassen, was diesem technisch keine Probleme bereiten sollte, da er vermutlich auch selbst Zonen bedient. Damit kann man diese beiden als NS-Einträge verwenden, so dass er Primary gar nicht gefragt wird. Das ermöglicht, die Firewall entsprechend scharf einzustellen.

DNS ist sicherheitsempfindlich. Gefälschte Datenbanken können es einem Eindringling z.B. ermöglichen, auf NFS- Shares zuzugreifen, wenn er die entsprechende in-addr.arpa.-Zone modifiziert. Auch mit dem Verändern der lokalen DNS-Server kann er Sicherheitslücken erzeugen, so z.B. Anfragen zu einer anderen Maschine umleiten, und so z.B. Passwörter stehlen, von einer Vielzahl von Denial-Of-Service-Attacks ganz zu schweigen.

Betreibt man selbst Secondaries, so sollten auf diesen Maschinen möglichst wenig andere Dienste laufen (das gilt selbstverständlich ganz besonders für Primaries!).

Secondaries sollten "check-names warn;" verwenden, um einen Ausfall durch einen Tippfehler oder ähnliches möglichst zu verhindern.

## 29 Zugriffsbeschränkungen konfigurieren

Nach Möglichkeit sollte der Zugriff so weit wie möglich beschränkt werden. Verwendet man kein dynamisches Update, so sollte man die option "allow-update { none; };" verwenden, ansonsten die IP-Adresse von jeder berechtigten Maschine aufzählen. Steht diese Option im "options {};"-Block, so gilt sie für alle Zones. Hat eine Zone einen eigenen Eintrag, so überschreibt dieser den globalen im "Options {};"-Block stehenden.

Ein Primary, der nirgendwo als "NS"-Record auftritt, wird normalerweise nie Anfragen erhalten, außer Zonetransfers von seinen Secondaries. Deshalb kann man in diesem Falle die Option "allow-query { none; };" setzen, diese darf ebenfalls im "options {};"-Block stehen, und damit global gelten.

Sehr häufig erlauben die Server Zonetransfers nur ihren Secondaries. Damit erlaubt auch ein Secondary (der von keinem als Primary verwendet wird, theoretisch lassen sich die Server aber **kaskadieren**) überhaupt keine Zonetransfers, höchstens zu ein, zwei Maschinen zu Testzwecken (wo z.B. `dnswalk` läuft, das Zonetransfers benötigt). Das konfiguriert man mit der Option "allow-transfer { none; };", bzw. bei Primaries durch Aufzählen der IP-Adressen der Secondaries (und evtl. Testhosts), z.B. "allow-transfer { 1.2.3.4; 1.2.3.5; };". Anstatt einzelner Adressen können selbstverständlich auch Netzwerke angegeben werden.

`bind` (ab Version 8) unterstützt auch **ACLs** (Access Control Lists), was die Übersicht erhöht. Hier kann man symbolische Namen für IP Adressen und Netzwerke vergeben, ein Beispiel:

```
/etc/named.conf

acl fremde_clients { !1.2.3/24; any; }; // alle, außer 1.2.3.0/24
acl locals { 1.2.4.0/24; }; // eigenes Netz
acl transfer { 1.2.10.53; 1.2.11.53; }; // Secondaries
acl clients { locals; transfer; }; // Resolver
acl dhcp { 1.2.4.254; }; // DHCP-Server

allow-transfer { transfer; };
allow-query { clients; };
allow-update { dhcp; };
```

Das ist dann besser lesbar.

Dabei müssen die ACLs natürlich vor deren Verwendung definiert werden. Sollen diese bereits im "options {};"-Block verwendet werden, müssen diese davor (also außerhalb dieses Blockes) definiert werden. Finden die ACLs in den "zone {};"-Blöcken Anwendung, so können nach dem "options {};"-Block (und vorzugsweise vor dem ersten "-zone {};"-Block) definiert werden. Die Struktur sähe z.B. so aus:

/etc/named.conf

```
acl transfer { 1.2.10.53; 1.2.11.53; }; // Secondaries
options {
    . . . .
    allow-transfer { transfer; };
};
acl locals { 192.168.0.0/16; }; // eigenes Netz
zone "my.play.net.de" {
    . . . .
    allow-query { locals; };
};
```

Damit könnte man dann eventuell eine interne, private Zone für private IP-Adressen auf dem normalen DNS-Server konfigurieren, aber den Zugriff aus dem Internet verbieten. Ansonsten kann es zu Störungen fremder Clients führen, da Namen u.U. falsch aufgelöst werden könnten.

## 30 Betrieb hinter einer Firewall

Hierbei gibt es zwei zu trennenden Dinge: DNS-Server, die Clientanfragen bearbeiten, und Autoritative-Server. Man kann diese auf getrennten Maschinen unterbringen, denn DNS-Server, die Clientanfragen beantworten, sind von Denial-Of-Service Attacken nicht so stark betroffen, wie Autoritative-Server. Mit der Option

```
query-source address * port 53;
```

kann man `bind` anweisen, alle Anfragen von Port 53 aus zu machen. Damit kann man die Firewall für die hohen Ports (>1023) auch noch sperren. Zusätzlich kann man auch die Zieladressen begrenzen, wenn man den Server "forward-only;" konfiguriert. Dann gibt man ihm z.B. drei Forwarder, und erlaubt nur Verbindungen zu diesen. Diese Verbindungsregeln müssen für UDP und TCP gelten, da `bind` bei Anfragen, die nicht mehr in ein einzelnes UDP-Paket passen, TCP verwendet. Selbstverständlich sollte die Firewall Regelverstöße loggen.

Breibt man einen Primary, und läßt die Secondaries vom ISP machen, so muß mit diesem abgesprochen werden, von welchen Maschinen aus welche Art von Zugriff zu erlauben ist. Einige verwenden z.B. `ping` für Tests, außerdem wird evtl. von einer anderen Maschine zu Testzwecken ein Zonentransfer probiert, bevor der Eintrag in den "richtigen" Server gemacht wird. Erhält man ungenügende Auskunft, so erstellt man eine Firewallregel, die alle Anfragen durchläßt, aber dabei loggen (Es reicht aus, Pakete mit gesetztem SYN-Bit zu loggen, allerdings macht das in der Praxis selten Sinn, da zu verbotenen Ports sowieso nur SYN-Pakete kommen werden, und der Aufwand für die zusätzlichen Regeln [die für TCP] kaum Nutzen bringt, sofern man keine "ACCEPTS" loggt). Dann kann man das Logfile analysieren, und erkennt, welche Maschinen Daten austauschen.

Leider kann man DNS nicht durch den `rinetd` redirecten, da dieser kein UDP unterstützt.

Da mindestens ein Slave außerhalb des Netzes steht, muß also mindestens dieser durch die Firewall dürfen. Um das testen zu können, sollte immer ein zusätzlicher Host erlaubt werden, also sowohl durch die Firewall dürfen, als auch einen Zonentransfer machen dürfen. Dann kann man `nslookup` auf diesem Host verwenden, um mit `ls <Zonenname>` einen Test-Zonentransfer durchzuführen.

Beim ersten Zugriff der Secondaries sollte man immer das Firewall- Log im Auge behalten, da die Provider gern einen Host, der Zugriff benötigt, vergessen. Das ist z.B. die Testmaschine, an dem die Zonen geprüft werden. So sieht man das Problem sofort, und kann eine entsprechende Regel evtl. schnell einfügen.

## 31 Classless Routing (CIDR)

Bei der klassischen Struktur des Internet gab es als kleinstes Netzwerk ein "Class C"-Netz mit 8 Bit Hostanteil, also für 254 Maschinen. Da viele kleinere Firmen existieren, wäre das eine große Verschwendung von IP-Adressen, deshalb vergibt man auch Teilnetze davon (z.B. mit 4-7 Bit Hostanteil, also /25 bis /29 Netze mit 6 bis 126 nutzbaren Adressen). Diese Technik nennt man **CIDR** (Classless Internet Domain Routing - Klassenloses Internetrouting), weil man nicht mehr an Klasse-A/B/C-Netze gebunden ist, und ein klassisches A-Netz z.B. wie 256 B-Netze routen und vergeben kann.

Dabei tut sich dann ein Problem auf: die Auflösung der IP-Adressen in Namen in der in-addr.arpa Zone. Da Delegation ja nur an den "Punkten", also an der Bytegrenze einer IP-Adresse möglich ist, muß das C-Netz, das zu verschiedenen Teilen von verschiedenen Firmen verwendet wird, vom Provider verwaltet werden, da erstmal keine Delegation möglich ist. Das ist dann unpraktisch, da eine Firma jede Veränderung über diesen Umweg durchführen muß.

Mit einem Trick, der auf ein Newsposting von *G.A. Herrmannsfeldt* zurückgeht (und inzwischen in *RFC 2317* beschrieben ist), lassen sich die Adressen eines solchen Teilnetzes dann aber doch delegieren. Die Idee dabei ist, dass in der eigentlichen in-addr.arpa die Adressen CNAME-Records sind (also Aliasnamen), die auf andere Zonen zeigen. Diese anderen Zonen lassen sich dann "delegieren", bzw. werden dort und dafür die entsprechenden Nameserver eingesetzt. Im klassischen Falle sähe die unpraktische Zone beim Provider ja so aus (zu den Netzen 192.168.2.0/30, 192.168.2.4/30 usw.):

2.168.192.zone			
\$TTL 1d			
2.168.192.in-addr.arpa. IN SOA pri-ns.selflinux.de. admin.selflinux.de. (			
			2000012501 ; Serial (Seriennummer)
			3H ; Refresh (Aktualisierung)
			1H ; Retry (neuer Versuch)
			1M ; Expire (ungültig nach)
			1D ) ; min. TTL (mindeste Gültigkeit)
	IN NS		ns1.selflinux.de.
	IN NS		ns2.selflinux.de.
1	IN PTR		lisa.simpson.net.
2	IN PTR		bart.simpson.net.
5	IN PTR		ernie.sesam.org.
6	IN PTR		bert.sesam.org.

usw. Nun möchte man das also über CNAMEs in andere Zonen zeigen lassen. Als Namen für diese Zone könnte man die Netzwerknummer verwenden (also 0.2.168.192.in-addr.arpa und [kurz] "4" - die Telekom macht das z.B. so), oder auch noch die Broadcastadressen dazunehmen (also "0-3"), oder die Netzmaske ("0-3", oder auch "0/30", was genau genommen nicht verboten ist, da es sich hierbei nicht um einen Hostnamen handelt, wäre der "/" erlaubt), also erhält man z.B. (nur diese Records):

2.168.192.zone			
1	IN	CNAME	1.0-3.2.168.192.in-addr.arpa.
2	IN	CNAME	2.0-3.2.168.192.in-addr.arpa.
5	IN	CNAME	5.4-8.2.168.192.in-addr.arpa.
6	IN	CNAME	6.4-8.2.168.192.in-addr.arpa.

Der Zone 0-3.2.168.192.in-addr.arpa. müssen dann natürlich noch NS-RRs hinzugefügt werden. Diese müssen in der zu "0-3" übergeordneten Zone eingetragen werden - also in der, in der auch die CNAMEs stehen. Das ist günstig, man erhält zusätzlich z.B.:

2.168.192.zone	
0-3.2.168.192.in-addr.arpa.	IN NS ns1.simpson.net.
0-3.2.168.192.in-addr.arpa.	IN NS ns2.simpson.net.

In diesen delegierten Zonen werden dann vom DNS-Admin von simpson.net die PTR Records untergebracht. Das sieht dann z.B. so aus:

0-3.2.168.192.zone	
\$TTL 1d	
0-3.2.168.192.in-addr.arpa. IN SOA pri-ns.simpson.net. admin.simpson.net. (	
2000012501	; Serial (Seriennummer)
3H	; Refresh (Aktualisierung)
1H	; Retry (neuer Versuch)
1M	; Expire (ungültig nach)
1D )	; min. TTL (Mindestgültigkeit)
IN NS ns1.simpson.net.	
IN NS ns2.simpson.net.	
1	IN PTR lisa.simpson.net.
2	IN PTR bart.simpson.net.

Es ist auch nicht zwingend notwendig, die "erfundenen" neuen Zonen unter die in-addr.arpa.-Zone zu legen, man könnte auch 0-3.simpson.net. verwenden, oder gar arpa.simpson.net. Das muß je nach Fall entschieden werden. Diese Entscheidung liegt beim übergeordneten Provider.

Leider kann man dieses Verfahren nicht mehrfach anwenden, um weiter zu delegieren, da in diesem Fall ein CNAME auf einen CNAME zeigt, was nicht sein soll (und was damit zu Problemen führen kann).

## 32 Umgang mit Störungen

Schäden durch DNS-Serverausfälle lassen sich durch redundante Systeme weitgehend vermeiden. Clients sind so zu konfigurieren, dass sie mehrere Server kennen. Fällt ein solcher Server aus (der ja ein caching-only Nameserver sein kann, also einer, der für keine Zone authoritative ist), wechselt der Resolver einfach auf einen anderen, bis auf etwas Zeitverlust kein weiterer Nachteil. Die Reihenfolge der Server ist von Client zu Client zu mischen, um Lastverteilung zu erreichen.

Es sind ausreichend viele Secondaries bereitzustellen, ansonsten wird der Betrieb u.U. durch Überlast gestört. Verwenden lokale Clients andere DNS-Server, so wird das u.U. spät bemerkt. Fällt ein Secondary aus, stellt das kurzfristig kein allzu großes Problem dar, solange noch mindestens ein Secondary verfügbar ist. Dabei muß vermieden werden, dass in diesem Falle der oder die verbleibenden Secondaries überlastet werden.

Fällt ein Primary aus, und ist dieser nur Primary, aber keiner mit einem NS-RR aufgeführter (auch **hidden primary**; genannt), so verläuft der Ausfall vollkommen transparent, bis die Secondaries die Zone expiren (also wegwerfen), was so um die zwei bis sechs Wochen dauern sollte (je nach Konfiguration der SOA-Records der Zone). Nach dieser Zeit ist die Zone jedoch vollkommen unbekannt! In der Praxis ist das jedoch mehr als genug Zeit, um einen neuen Server aufzusetzen, ein Backup einzuspielen, und so notfalls einfach den kompletten Server zu wechseln.

Da DNS auf verteilten Datenbanken beruht, muß bei einer Kontrolle bzw. beim Debugging beachtet werden, welcher Server (welcher Secondary, welcher Forwarder etc.) welche Antwort gibt. So kann es z.B. sein, dass ein Secondary fehlerhaft arbeitet. Ein Test kann aber die richtige Antwort liefern, falls dieser zufällig nicht gefragt wird. Diese Antwort wird dann evtl. noch von einem Forwarder gecached. Deshalb sollte man bei Tests die Secondaries einen nach dem anderen direkt fragen. Widersprechen sich z.B. zwei Secondaries, können die Fehler diffus und versteckt auftreten.

Haben sich die Zonendateien in letzter Zeit geändert, ist es auch möglich, dass ein Secondary den Transfer noch nicht durchführen konnte. Deshalb muß als erstes geprüft werden, ob die SOA-Serial überhaupt die aktuelle ist. Eventuell gibt es Probleme mit der Notification (wenn alles funktioniert, sollten die aktualisierten Zonen in wenigen Minuten den Secondaries bekannt sein).

Es gibt auch Fehler, die eigentlich in anderen Zonen liegen, als in der sie bemerkt werden. So ist es zum Beispiel denkbar, dass ein NS-Record fehlerhaft scheint, in Wirklichkeit aber der dazugehörige A-Record aus der Zone, zu der der DNS-Server gehört, fehlerhaft ist oder nicht existiert. Diese Zone wird dann eventuell auch noch von anderen verwaltet...

## 33 Root-Rechte

Häufig wird BIND unter root-Rechten gestartet. Das ist natürlich ein Sicherheitsrisiko und sollte vermieden werden. Bei aktuellen Versionen ist das sehr einfach. Zunächst müssen eine eigene Gruppe und ein eigener Benutzer eingerichtet werden. Unter Linux kann man diese ganz einfach mit den folgenden beiden Kommandos anlegen:

```
root@linux / # groupadd named
root@linux / # useradd -d /var/named -g named named
```

Nun muß das Verzeichnis `/var/named` an diesen Benutzer "verschenkt" werden:

```
root@linux / # chown -R named:named /var/named
```

Eigentlich besteht kein Grund, eigene, primäre Zonenfiles (also die, für die der Server authorativ und kein Secondary ist) zu verschenken. `bind` muß aber in der Lage sein, sekundäre Zonefiles zu schreiben, nachdem sie vom Primary geladen wurden. Dazu kann man aber auch ein Unterverzeichnis verwenden, und nur dieses schreibbar machen. Dann kann im Falle des Mißbrauches der Benutzererkennung (z.B. durch einen erfolgreichen Angriff) kein primäres Zonefile verändert werden.

Nun muß der Startaufruf um zwei Parameter erweitert werden, damit bind weiß, das er unter anderen Benutzerrechten laufen soll:

```
root@linux / # /usr/sbin/named -u named -g named
```

Nach der Initialisierung werden dann die root-Rechte abgelegt. Beim Start sind diese natürlich noch kurz erforderlich, da sonst der Port 53 nicht verwendet werden könnte (privilegierte Ports dürfen nur von root verwendet werden).

## 34 Chroot-Umgebung

Um es noch sicherer zu bekommen, kann man `bind` auch in einer `chroot`-Umgebung laufen lassen. Das macht allerdings etwas mehr Arbeit, denn alle von `bind` benötigten Dateien müssen dann unterhalb dieses `chroot`-Verzeichnisses liegen. Dazu sind dann einige Schritte notwendig. Vorallem muß `bind` als eigener User laufen, da `chroot` mit Root-Rechten nicht sinnvoll funktioniert.

Zunächst muß eine minimale Verzeichnisstruktur erzeugt werden, die `bind` verwenden kann. Sie kann z.B. unter `/var/named.chroot` erzeugt werden. Dazu folgendes Kommando (als root!):

```
root@linux / # mkdir -p /var/named.chroot/etc
root@linux / # mkdir -p /var/named.chroot/var/named
root@linux / # mkdir -p /var/named.chroot/var/run
root@linux / # mkdir -p /var/named.chroot/bin
root@linux / # mkdir -p /var/named.chroot/usr/sbin
root@linux / # mkdir -p /var/named.chroot/dev
root@linux / # mkdir -p /var/named.chroot/lib
```

Nun müssen die Konfigurationsdatei und die Datenbanken kopiert werden:

```
root@linux / # cp -p /etc/named.conf /var/named.chroot/etc/
root@linux / # cp -pR /var/named /var/named.chroot/var
```

Nun sind die Rechte der Dateien zu korrigieren:

```
root@linux / # chown -R root:root /var/named.chroot/
root@linux / # chmod 755 /var/named.chroot
root@linux / # chmod 755 /var/named.chroot/*
root@linux / # chown -R named:named /var/named.chroot/var/named/
root@linux / # chown -R named:named /var/named.chroot/var/run/
root@linux / # chmod 644 /var/named.chroot/etc/*
root@linux / # chmod 755 /var/named.chroot/lib/*
root@linux / # chmod 755 /var/named.chroot/usr/sbin
root@linux / # chmod 755 /var/named.chroot/usr/sbin/*
```

Normalerweise verwendet man dynamisch gelinkte Binärdateien, und so müssen die Bibliotheken kopiert werden. Zuerst muß man schauen, welche Libs benötigt werden:

```
root@linux / # ldd `which named`
```

Eine Ausgabe könnte sein:

```
libc.so.6 => /lib/libc.so.6 (0x4001d000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Es kann bei anderen Unix-Derivaten sein, dass das Kommando `ldd` oder `which` nicht verfügbar ist. Dann muß ein anderer Weg gefunden werden.

Diese Dateien werden kopiert:

```
root@linux / # cp -p /lib/libc.so.6 /var/named.chroot/lib/  
root@linux / # cp -p /lib/ld-linux.so.2 /var/named.chroot/lib/
```

Dann benötigt BIND die Gerätedatei `/dev/null`. Hat man ein modernes `/bin/cp`, so reicht ein:

```
root@linux / # cp -a /dev/null /var/named.chroot/dev/
```

andernfalls muß man das Gerät neu anlegen:

```
root@linux / # mknod /var/named.chroot/dev/null c 1 3
```

(Die konkreten Parameter können wiederum bei anderen Unix-Derivaten abweichen).

BIND benötigt die Datei `/etc/localtime`:

```
root@linux / # cp /etc/localtime /var/named.chroot/etc/
```

und natürlich seine eigenen `/etc/passwd`-Einträge. Diese können z.B. mit folgenden Einzeilern erzeugt werden:

```
root@linux / # cat /etc/passwd | grep 'named' >  
/var/named.chroot/etc/passwd  
root@linux / # cat /etc/group | grep 'named' >  
/var/named.chroot/etc/group
```

Wenn `bind` über `syslogd(8)` loggt, und nicht nur über eigene logfiles, muß hier noch weiter angepaßt werden. Normalerweise greift BIND auf `/dev/log` zu. Diese Datei ist nun aber nicht mehr erreichbar. `syslogd` muß also eine Datei zusätzlich verwenden, die erreichbar ist. Ein halbwegs modernes `syslogd` kennt dazu den Parameter `"-a"`, der nun beim Start des Syslogdeamons angegeben werden muß. Dazu muß u.U. das Startscript angepaßt werden. Bei einer SuSE-Distribution kann man aber auch den Parameter `SYSLOGD_PARAMS` in `/etc/rc.config` verwenden:

```
/etc/rc.config
```

```
SYSLOGD_PARAMS="-a /var/named.chroot/dev/log"
```

Sonst muß der `"-a"`-Parameter an der entsprechenden Stelle im Startscript hinzugefügt werden. Danach ist `syslogd` neu zu starten. dabei sollte `syslogd` nun diese Datei erzeugt haben. Leider neigen scheinbar einige `syslogd`-Versionen dazu, hier Bugs zu besitzen, und nur noch diesen Socket auszuwerten oder dergleichen. Ein Update sollte dem abhelfen.

Soll `bind` nicht über `syslogd` loggen, sondern in Dateien schreiben, kann folgendes konfiguriert werden:

```
/etc/named.conf
```

```
logging {
    channel file_log {
        file "named.log";
        severity info;
    };
    channel file_debug {
        file "named.run";
        severity dynamic;
    };
    category default {
        file_log; file_debug;
    };
};
```

Nun müssen die Programme selbst kopiert werden, mindestens `named` und `named-xfer` (mit `which named` kann herausgefunden werden, wo sich diese befinden). Dabei muß beachtet werden, dass der Pfad zu `named-xfer` "hardcoded" im Binärprogramm steht. Dieser Pfad muß dann um den `chroot`-Pfad erweitert werden, damit das Programm von `bind` gefunden wird. Liegen `named` und `named-xfer` z.B. in `/usr/sbin` (wie bei SuSE), so müssen sie nach `/var/named.chroot/usr/sbin` kopiert werden:

```
root@linux / # cp /usr/sbin/named* /var/named.chroot/usr/sbin/
```

Man kann natürlich auch in ein anderes Verzeichnis kopieren, muß dann aber in der Konfigurationsdatei den Pfad entsprechend angeben, z.B.:

```
/etc/named.conf
```

```
options {
    ...
    named-xfer "/bin/named-xfer";
};
```

Dabei ist wieder zu beachten, dass die benötigten Libs geladen werden können. Der hier angegebene Pfad ist natürlich um das `Chroot`-Verzeichnis gekürzt anzugeben, da `BIND` dann von dieser nichts mehr sieht, also unter `/bin` etwas anderes versteht, als ein nicht `ge-chrooteter` Prozeß.

Der Startaufruf für `BIND` muß nun nur noch um den Parameter `-t` erweitert werden:

```
root@linux / # /usr/sbin/named -u named -g named -t /var/named.chroot
```

Nun sollte überprüft werden, ob Anfragen noch richtig beantwortet werden, und ob ein Zonentransfer funktioniert.

Da (ein älteres) `ndc` das `pid`-File in `/var/run` erwartet, und nicht in `/var/named.chroot/var/run`, kann hier ein Symlink helfen, der vom Startscript nach dem Starten jeweils erzeugt wird:

```
root@linux / # ln -s /var/named.chroot/var/run/named.pid /var/run/
```

Beim Beenden kann der Link dann einfach gelöscht werden:

```
root@linux / # rm -f /var/run/named.pid
```

Es ist zu beachten, dass `ndc restart` und `ndc start` nicht mehr verwendet werden sollten, da hier die Parameter aus unserem Startscript nicht verwendet werden! Dann wird `bind` unter root-Rechten ohne `chroot` ausgeführt!

Modernere `ndc`-Versionen handhaben die Verbindung anders. Es wird ein **control channel** verwendet, defaultmäßig ist das `/var/run/ndc`. Das kann über Parameter geändert werden. Ein modernes `ndc` kann aber auch über Parameter dazu veranlaßt werden, ein pid-File zu verwenden. Diese Defaults können beim Compilieren allerdings geändert werden. Sonst lautet der Aufruf von `ndc` wie folgt:

```
root@linux / # ndc -c /var/named.chroot/var/run/ndc
```

Es ist wohl eine gute Idee, hierauf einen Shell-"Alias" zu setzen, z.B. in `~/.profile`:

```
~/.profile
```

```
alias ndc='ndc -c /var/named.chroot/var/run/ndc'
```

Es sollte mindestens ein `reload` probiert werden, um zu sehen, ob z.B. `bind` jetzt noch seine Konfigurationsdatei lesen kann. Beim ersten Start funktioniert das immer, da vor dem Setzen der `chroot` Umgebung noch root-Rechte vorhanden sind, was bei `reload` ja nicht mehr der Fall ist. Gegebenenfalls sind die Rechte zu kontrollieren. Hierbei gilt, `bind` darf ruhig viel (alles) lesen können, aber nur die Slave-Zones schreiben (also darf dem User nur wenig gehören - nur das Slave-Zonen-Verzeichnis).

## 35 Versionen

Da `bind` leider immer wieder Sicherheitslücken enthält, ist es ein beliebter Angriffspunkt für Angreifer. Da zu vielen `bind`-Versionen Sicherheitslücken bekannt sind, kann eine Angreiferin auf die Idee kommen, einfach die Version abzufragen, und dann kann sie die nun bekannten Sicherheitslücken ausnutzen.

Deshalb mag es günstig sein, die Versionsnummer nicht zu verraten, um der Angreiferin das Leben etwas schwerer zu machen.

Die Version läßt sich erfahren, wenn man einen `BIND`-Server nach "version.bind", Type `TXT`, in der Klasse `CHAOS` fragt. Das geschieht z.B. mit folgendem Kommando:

```
root@linux / # dig @<server> version.bind TXT CHAOS
```

Eine Antwort sieht dann (gekürzt) so aus:

```
VERSION.BIND.          0S CHAOS TXT      "8.1.2"
```

Man kann dazu natürlich auch `nslookup` verwenden:

```
root@linux / # nslookup -class=CHAOS -query=txt version.bind.
```

Neuere `BIND`-Versionen kennen eine Konfigurationsoption, um die dabei angezeigte Zeichenkette zu setzen. Man kann sich damit dann eine Version `10.100.1000` basteln, oder auf etwas ganz anderes setzen. Das kann dann so aussehen:

```
/etc/named.conf
```

```
options {
    ...
    version "[Secured]";
};
```

Dann wird mit dem Kommando das auch angezeigt:

```
VERSION.BIND.          0S CHAOS TXT      "[Secured]"
```

Und Rückschlüsse auf die Version sind so nicht mehr möglich. Diese Änderung kann natürlich auch gemacht werden, in dem der RR "version.bind" in einer Zone gesetzt wird. Dabei können dann, wie für jede andere Zone auch, ACL-Berechtigungen konfiguriert werden. Diese Zone wird so konfiguriert, wie auch die Zonen der normalen Namen der Klasse `IN`. Dazu benötigt man z.B. folgenden Eintrag in der Konfigurationsdatei:

/etc/named.conf

```
zone "bind" CHAOS {
    type master;
    file "bind.zone";
    allow-query { none; }; //Zugriff verweigern
    allow-transfer { none; }; //damit AXFR auch nicht geht.
};
```

Natürlich kann man auch Ausgewählten den Zugriff erlauben. Nun muß noch die Datenbank selbst angelegt werden. Sie kann folgenden Inhalt haben:

bind.zone

```
bind. CHAOS SOA pri-ns.selflinux.de. admin.selflinux.de. (
    2000012501 ; Serial (Seriennummer)
    3H        ; Refresh (Aktualisierung)
    1H        ; Retry (neuer Versuch)
    100D      ; Expire (ungültig nach)
    1D )      ; min. TTL (Mindestgültigkeit)

    CHAOS    NS        dns
version CHAOS    TXT    "[Secured]"
```

## 36 Wachstum

Wichtig ist vor allem, Störungen schnell zu erkennen und so beseitigen zu können. Auch die Auslastung in Spitzenzeiten ist zu überwachen. Wird diese zu hoch, kann man über eine Delegation von Teilen der Domain nachdenken, oder weitere Secondaries einsetzen. Sind cachende Server überlastet, so ist es aufwendiger, neue einzusetzen, da diese ja bei den Clients konfiguriert werden müssen. Eventuell ist aber die Hardware ausreichend erweiterbar (vielleicht hilft auch erstmal ein Hauptspeicherausbau).

Um einen reibungslosen Betrieb zu erhalten, ist es wichtig, den Überblick zu haben. Deshalb sollte man sich die Zeit nehmen, Adresspläne zu pflegen, Topologiediagramme aktuell zu halten und Änderungen mit anderen Beteiligten zu synchronisieren und abzusprechen. Normalerweise sollte ein DNS-Admin wissen, wer für eine bestimmte Maschine verantwortlich ist, und der weiß, wo sie physikalisch steht. Im Idealfall erfährt der DNS-Admin auch, wenn Maschinen aus dem Betrieb genommen werden. Leider werden dessen IP-Adressen teilweise einfach stillschweigend für neue Maschinen verwendet. Erstmal kein Problem. Aber in drei Jahren vielleicht...

## 37 Keep it running

Dieser goldene Satz gilt natürlich auch für den DNS-Administrator. Lläuft **BIND** einmal, so muß mit nur wenig Wartungsaufwand gerechnet werden. Hin und wieder sollte man die Root-Server Einträge erneuern, da sich deren IP-Adressen mit der Zeit ändern können. Sicherheitshinweise sollten verfolgt werden, und so ist evtl. die Software zu updaten. Routineprüfungen schaden selbstverständlich nicht, insbesondere ein Beobachten der Logfiles (dazu verwendet man am besten ein Tool, wie **logsurfer** oder **logmail**).