The pitfalls of DNS

# DNS SUBTLETIES

**DNS is a distributed system that handles the correspondence between hostnames and IP addresses. In this article Wednesday White aims to discuss some of the more interesting cases that you may fall afoul of once you've got a basic DNS implementation up running**

## Alternative DNS servers

It's not compulsory to use the ISC's BIND package for DNS, and not everyone does; even if you do use BIND, you have a choice between the more stable and better understood version 8, or the relatively new version 9. Version 9 introduces some ingenious new features, some of which I will discuss below, but is probably more prone to new security holes being found and to general instability. I wouldn't suggest using version 9 unless you need one of these new features.

The other viable alternative is a package by Dan Bernstein (DJB) called "djbdns", available from *http://cr.yp.to/djbdns.html*. Djbdns is free of cost, but the license is not Open Source (although the source is open for inspection), meaning your GNU/Linux distribution probably does not include it; a more serious drawback of djbdns is that it assumes that you wish to organise your systems exactly as DJB would; and of course article authors will persist in being awkward and discussing everything in terms of BIND.

However, djbdns is believed to be extremely secure; at this time, no security holes in it have been exploited, and plenty of people have been looking – DJB offers a reward of $500 for finding one. BIND, by comparison, has been compromised all too often. If you are paranoid and want to run a DNS server that provides a service to the whole world or to possibly malicious people, djbdns may be for you. Djbdns also offers superior performance, but that's unlikely to be an issue except for the largest of sites.

The most ingenious feature of djbdns is that it divides up the features of a DNS implementation – caching, normal queries, answering zone transfers – into separate programs, making it easy to only provide the functionality you require on a particular server. Microsoft's Windows NT does also provide a DNS server, but I will not be discussing it here.

## More about security

Don't forget to subscribe to a relevant mailing list for security alerts – probably the one run by your chosen GNU/Linux distributor. Ensure you are running up-to-date versions of your software; 8.1.3 or 9.2.0 for BIND, 1.05 for djbdns at the time of writing.

It is well worth disabling zone transfers except from approved machines (with the allow-transfer statement, when using BIND); this should normally only be those machines that slave zones off a particular DNS server. This will prevent the black hats from grabbing a complete copy of your zone and looking through it for attractive targets, and (possibly accidental) DoS attacks on your server under with a series of zone transfer requests. If you run a "hidden primary" configuration, you may be able to disallow all requests to that server except from its slaves, not just zone transfers.

Do firewall off port 53 except to servers that actually provide a DNS service to the outside world; don't make the mistake of permitting only UDP port 53 because 'only zone transfers use TCP' – any reply over a certain length will use TCP port 53.

Consider also filtering outgoing port 53; you'll want to ensure that outgoing requests come only from the server or servers you want them to, especially if you're running a multiple horizon set-up – a set-up where the same domain has different data for internal users and external queries, which is very common if you don't want random people to know all the names of your internal machines. However, this is only appropriate if your internal machines' IP addresses can never be used on the public Internet (owing to some kind of NAT arrangement, perhaps using the RFC 1918 reserved ranges) – if their IP addresses are visible, they ought to have names, too!

## Reliability

Various system-monitoring scripts exist which can monitor several DNS servers and check they are all still answering queries; however, when setting up

## Multiple DNS servers

If you're running DNS in anything other than your home, pretty soon you're going to want to have more than one DNS server. But how many? In medium to large sized organisations, DNS servers can serve many functions. You'll want lightweight caching-only servers over a large network, to provide low latency answers to users; you'll also have nameservers that are connected to the public Internet, both to pass requests out from internal users and to provide information about the domain or domains you run yourself – and you probably want these to be separate machines, since the machines that accept requests from the outside world are necessarily more of a security liability and will ideally be placed in a DMZ. If you run a multiple horizon set-up, you'll need a second set of nameservers that provide the internal view of your domains.

such a thing, it's all too easy to ensure that the failure of a single monitoring machine or of your mail system completely disables alerting! Be careful. It's not much good having several DNS servers if they are all on the same network subnet where the failure of a single router or switch can take them all out; try to ensure that all your DNS servers could only be rendered inaccessible if the network was completely unusable. If your organisation is large enough to have more than one route to the Internet, try to ensure that your DNS architecture has at least one server using each one.

Conversely hardware for DNS servers does not need to be hugely expensive – although shelling out a little more is often worthwhile. The DNS is designed so that at every stage of the process, systems can have a choice of three or more servers to query; if you have avoided the network problems above, you will survive the failure of any particular server. However, you should ensure that you have copies of the configuration for each DNS server you possess in a number of places; then, when a particular machine suffers a terminal hardware failure, you can very easily produce another system with the same configuration to replace it – particularly if you use Free operating systems on cheap hardware, and can hence readily have spare machines with an OS installed ready to be used at any time.

## Hidden primary

A "hidden primary" configuration is one where the master for a particular zone is not actually mentioned in the NS records for that zone at all; instead, a set of machines all of which slave the zone off it are mentioned. This has some advantages; the hidden primary never receives any DNS requests except approved zone transfers (no-one knows its name, and it need not even be willing to answer them), so will not be heavily loaded even if you run all your zones off it; and if you make an error editing a zone file and the nameserver refuses to load it, none of the nameservers that anyone actually uses will be refusing queries because they have no data for that zone. The benefit of concentrating all your zone files in one place without performance worries is considerable, and should not be overlooked.

## Multiple horizon

Traditionally multiple horizon setups have required two complete sets of nameservers, which is a pain. BIND 9 added the "split view" facility that, with appropriate configuration, allows you to load two different sets of zone files and answer requests based on the IP address of the calling client. In a hidden primary setup, the primary can use split view – with reduced security worries, since although it will run BIND 9 it need not accept DNS traffic from random machines at all – to serve both internal and external zone files to its slaves, permitting you to concentrate all editing on one machine.

A simplification of multiple horizon setups is to use a separate domain for all your internal entries; if your world-facing domain is "example.com", ensure that all your internal machines are in "internal-example.com" (however, you should ensure you register this domain) – then your multiple horizon set-up need only ensure that your world-facing DNS servers believe they are authoritative for it and load an empty zone file for it.

## Reverse DNS

Reverse DNS is something that is traditionally messed up; but I'd encourage you to make a break with tradition and get it right! Very few people do; ISPs are some of the worst offenders, with plenty of Internet-accessible machines (usually routing hardware) lacking reverse entries.

If a given IP address is in use – if a computer has it assigned, or if any forward DNS entry resolves to that IP address – that IP address ought to have a reverse DNS entry; and that reverse entry ought to resolve to a name which can itself be looked up to yield the same IP address. Note that it's not a problem if elephant.example.com resolves to 192.168.53.76 and 192.168.53.76's reverse entry is rhino.example.com –

## Some common errors

Unfortunately there are more common errors than this; these are just some of the more awkward ones.

- The standards specify that an MX record – used for mail delivery – cannot point to a CNAME. Unfortunately, this usually appears to work OK, and so goes unfixed; nevertheless, it is a surprisingly awkward case for authors of mail transfer agents to get right, and should be eliminated. MX records should always point to A records.
- When editing a zonefile, leaving the trailing dot off a fully qualified domain name is an incredibly common error; this of course results in the zone being appended to the entry, producing absurdities like "reverse entry for 192.168.53.90 is snake.example.com.53.168.192.in-addr.arpa." It's easy to do; the answer is always to test an entry immediately after changing it and reloading the nameserver.
- A particular IP address can have multiple A records pointing to it; a common error is to fail to notice that a reverse record already exists when adding a second A record pointing to a given IP address, and add a second IP address, which causes the nameserver to reject the reverse zonefile. The simplest answer is to keep reverse zonefiles sorted – then the previous reverse entry will be obvious when you try to add the new bogus one.
- Failing to increment the serial number when editing a zone file causes remote nameservers not to think the zone has changed. Unfortunately, this is just a matter of training yourself not to forget – or using a tool like h2n that does it for you.

If your ISP is not sufficiently competent, they will tell you it can't be done

provided that rhino.example.com also resolves to 192.168.53.76.

The first problem that you will probably encounter is that your ISP is unable or unwilling to delegate the relevant reverse ranges to you. This is very common with bargain-basement operations that will sell you a domain and delegate you the forward zone, but find reverse DNS to be a mystery. Normally this is just a matter of persuasion, but it's more difficult in the case where your IP address range is not what used to be a class A,B or C network (for example when your subnet mask is not 255.0.0.0, 255.255.0.0, or 255.255.255.0).

Fundamentally, the design of the in-addr.arpa zone used for reverse DNS is intended only to deal with these cases, since it predates the CIDR system now in use. If your ISP is not sufficiently competent, they will tell you it can't be done.

How can you deal with this? It's detailed in RFC 2317; on your end it's simple enough. You insert zone file definitions starting like this;

```
zone "64/27.53.168.192.in-addr.arpa" {
```

into your named.conf – this one would be to deal with reverse entries in the 192.168.53.64/27 subnet, which contains 32 IP addresses. (Of course, the IP addresses here are from the RFC 1918 reserved ranges, and so would never be used on the global Internet.)

Your ISP – which controls 53.168.192.in-addr.arpa, we hope – delegates 64/27.53.168.192.in-addr.arpa to you with lines like this in the zonefile for 53.168.192.in-addr.arpa ;

```
64/27      NS   <your name server>
64/27      NS   <your other name server>
```

They also create one entry for each IP address in your subnet, like this;

```
64        CNAME  64.64/27.53.168.192.➚
in-addr.arpa.
65        CNAME  65.64/27.53.168.192.➚
in-addr.arpa.
(and so on for 30 more entries up to)
96        CNAME  96.64/27.53.168.192.➚
in-addr.arpa.
```

Of course this is a pain, but they only have to do this once and all these entries can be automatically generated. Now you can create entries in your zonefile for 64/27.53.168.192.in-addr.arpa like this one;

```
73        PTR   giraffe.example.com.
```

Now if someone looks up 73.53.168.192.in-addr.arpa they will find a CNAME to 73.64/27.53.168.192.in-addr.arpa ; they will find that 64/27.53.168.192.in-addr.arpa is delegated to you, and ask your name servers; and they will return the answer 'giraffe.example.com'.

## Alternatives to editing zone files

It's not really an alternative, but a lot of the pain of editing zone files can be alleviated by using a version control system such as GNU CVS. If you have more than one person editing zone files, I would go so far as to say that this is an absolute requirement.

Beyond that, the venerable h2n script transforms lists of hosts and IP addresses into correctly formed zone files; it can readily include other chunks of zonefile, for things you cannot describe as host-IP pairs (like MX records). It increments the serial number eliminating another common source of error. If you aren't doing anything overly complex, ensuring you edit lists of hosts and then run h2n on them can greatly simplify DNS maintenance.

If you want to get more sophisticated, you will end up writing your own Perl scripts to find free addresses, free up old addresses, check the correctness of zonefiles, make the coffee, and so forth. This can certainly be an interesting project (and provides for people who faint at the words 'text editor'), but is probably overkill unless you really are running a huge DNS set-up.

Some proprietary software vendors make "IP management" software; in my experience these are clunky, slow, painful to use, and do not provide even the most basic sanity checking. Steer clear.

## DNSSEC

This is perhaps the most significant improvement in Bind 9. The DNS is very vulnerable to 'spoofing' – insertion of bogus data into caches designed to misdirect traffic to the wrong machines.

A detailed discussion of DNSSEC would require another article, but essentially DNSSEC uses public key cryptography so that a zone can sign its subzones; hence, if I have a public key for "example.com" and I receive data for "animals.example.com", my nameserver can check that the source of data for "animals" has a public key signed by the owner of the private key for example.com, and hence that the data comes from an approved source. Ultimately, of course, the key signing "web" will come down from the root nameservers, so it will not be necessarily to trust any keys at all – the key for example.com will be signed with the key for .com, which will itself be verified by the root nameservers.

The BIND 9 Administrator's Manual contains a discussion of the necessary steps to get DNSSEC up and running; it's worth a look.