9

Dynamic DNS

With the appearance of dynamically allocated addresses in DHCP and dial-up protocols such as PPP, DNS has had some catching up to do. In April 1997 RFC 2136 was published, specifying a mechanism for dynamic updates of DNS, known as DNS-UPDATE. However, before getting into that I need to introduce some new terms.

Of RRsets

An RRset is the set of RRs associated with a name. In any zone there is at least one name which has several records associated with it. This is an extract from the penguin.bv zone as shown in Chapter 2, "DNS in Practice":

```
@       3600    SOA     ns.penguin.bv.  hostmaster.penguin.bv. (
                2000042924      ; serial
                86400           ; refresh, 24h
                7200            ; retry, 2h
                3600000         ; expire, 1000h
                172800          ; minimum, 2 days
                )
                NS      ns
                NS      ns.herring.bv.
                MX      10 mail
                MX      20 mail.herring.bv.
...
@               A       192.168.55.3
@               A       192.168.55.10
...
```

Associated with the name penguin.bv are one SOA record, two NS records, two MX records, and two A records. This makes one RRset. In RFC 2181, which updates RFC 1035, it is specified that all records in an RRset must have the same TTL. This RFC is listed as elective and this restriction is, in fact, not currently implemented in BIND 8. But acting like it is implemented in the case of dynamic zones might be a good idea, because in a dynamic zone the TTL of the records becomes very important. There is no mechanism to update cached RRs. They must be expired from the cache before any new value is requested from the zone servers.

Of Masters and Slaves

In the context of updates, what server to update becomes important, because there will be a need to know what server is the primary master. The primary master is named in the SOA record's first field. This field must not be used for the zone name. It must name the zone master server. Even if the server is otherwise unlisted, it must be named in the SOA record.

A slave server is a server that has a master. A master server is a server that acts as a master for a slave. The primary master is the zone origin server—it has no masters.

RFC 2136 specifies that, if a slave server receives an update request, the request shall be accepted and forwarded toward the primary master server. A slave server will not know who is authorized to update zones and so cannot check the authorization. It must store the update request and forward it. If the primary master is unavailable, it should forward it to any other master it knows. This allows the request to percolate through layers of protection that might be present between the zone updater and the primary master. That's the theory anyway. As of this writing (May 2000), BIND 8 does not implement this and the primary master must be present for an update to take hold. Even if you configure a slave to accept update requests and update the zone, the update will not find its way back to the primary master. Then, the next time a zone transfer from the primary master takes place, the update will be overwritten. So, the primary master must be available to the zone updater. This will be fixed in a future version of BIND.

Accepting and Doing Updates

The DNS Server

By default BIND DNS servers do not accept update requests. You must configure each zone on the server to accept updates from the appropriate clients. Those who are allowed to update zones can be defined in two ways. The easiest is to accept all update requests from a given host. This is not very secure and should only be contemplated within a firewall-protected network. Consider that it is relatively easy to spoof IP source addresses, and that anyone able to present the correct source address will be able to demolish the whole dynamic zone. Do not underestimate how simple this is; variations of spoofing have been used in a variety of attacks all over the Internet. So beware.

TSIG

Using TSIG updates is just a little bit harder than IP authentication and it is a lot more secure. *TSIG* is short for *transaction signature* and is a cryptographical signature that the server can check. If the signature is correct, the server knows that the update either came from the authorized client or from someone who has stolen the secret signing key. TSIG uses a mechanism called HMAC-MD5 to authenticate the sender and message content of the updates. HMAC is a mechanism for message authentication to be used in combination with a cryptographic hash routine, MD5 in this case. HMAC is described in RFC 2104; MD5 is described in RFC 1321. HMAC-MD5 is also specified for use in IP-SEC, and in RFC 2403 (a IP-SEC RFC) we find this nice summation:

> "HMAC is a secret key authentication algorithm. Data integrity and data origin authentication as provided by HMAC are dependent upon the scope of the distribution of the secret key. If only the source and destination know the HMAC key, this provides both data origin authentication and data integrity for packets sent between the two parties; if the HMAC is correct, this proves that it must have been added by the source."

HMAC-MD5, then, is a secret key, shared secret, or symmetric cryptography. This is different from public key cryptography, which is the kind of cryptography used for email. It is vital that the shared secret remains secret. If anyone manages to steal the key, they can trivially masquerade as you; then they don't even have to spoof their IP address. I'm not a cryptography expert and will not go farther into how this works, but see *Handbook of Applied Cryptography* or *Applied Cryptography* for more information on the subject of cryptography. TSIG is currently described in an Internet draft only; it is available at `http://www.ietf.org/internet-drafts/draft-ietf-dnsext-tsig-00.txt`. TSIG is based on HMAC-MD5.

These are also the properties we want for use with dynamic DNS.

The Dynamic Zone

An important thing to note is that a zone maintained dynamically cannot be maintained any other way. The primary master will maintain the zonefile automatically and BIND expects to be the sole updater of the file. Any effort to edit it manually will result in mayhem and confusion. Although, this does not stop you from maintaining the zone wholly with the dynamic DNS tools, and indeed you can do that.

But, due to this restriction and because they want to keep using the tools they know to maintain their normal zones, many sites set up separate zones for dynamic DNS.

The Penguin company has hired 16 new employees and wants to maintain their workstations in a dynamic zone. The company adds dyn.penguin.bv. This must be a separate zone, delegated in the normal way with NS records in the parent zone. In the penguin.bv zone, this is added to delegate the zone to the named servers:

```
dyn             NS ns
dyn             NS ns.herring.bv.
```

In named.conf dyn.penguin.bv is added in the normal way, less one detail—the allow-update ACL:

```
zone "dyn.penguin.bv" {
        type master;
        file "pz/dyn.penguin.bv";
        allow-update {
                192.168.55.2;
        };
};
```

This zone specifies which hosts are allowed to update the zone. In this case it is the name-server's own IP address. It will also act as a DHCP server so all updates will come from itself. To use TSIG authentication of updates, a key pair must first be generated:

```
# dnskeygen -H 512 -h -n ns.penguin.bv.
Generating 512 bit HMAC-MD5 Key for ns.penguin.bv.

Generated 512 bit Key for ns.penguin.bv. id=0 alg=157 flags=513
```

Two files were created: `Kns.penguin.bv.+157+00000.key` and `Kns.penguin.bv.+157+00000.private`. They both contain the same secret key, but in different formats. The `.key` file is in DNS zone file format. The key must be stolen for the thief to be able to sign as you would and thus be indistinguishable from you for the DNS server. Keep your key on a secure machine, and as with sensitive passwords, change it now and then—especially when key staff leaves you or whenever you have security incidents. Keeping your keys secure is not a matter to be taken lightly.

The public key must be entered in the named.conf file, or in a file it includes, and the file containing it must of course be read restricted. The audience able to read it should be miniscule.

```
key ns.penguin.bv. {
        algorithm hmac-md5;
        secret "XBzxlscP7rw3vfqF/yONoGNDQKMKgAcndBhKednuwlgqMc2rTVO
                jdGv4VqGyhj5uqW/uJlciyn/M045VFonxtQ==";
}
```

The key has been broken to fit in the book. In the file there should be no line break. The zone must be configured to allow for the holder of the key to change it:

```
zone "dyn.penguin.bv" {
        type master;
        file "pz/dyn.penguin.bv";
        allow-update {
                key ns.penguin.bv.;
        };
};
```

You can, of course, list several authorized keys and IP-numbers. I will get back to how the client uses the key in a second.

The next task is to seed the dynamic zone—to provide an initial zone file. It should include all the usual records for a zone, perhaps excepting any actual host records. You can add those later. Here then, is a seed file:

```
$TTL 1m
;
@       1m      SOA ns.penguin.bv. hostmaster.penguin.bv. (
                1               ; serial
                5m              ; refresh
                2m              ; retry
                6h              ; expire
                1m              ; minimum
                )
        1m      NS      ns.penguin.bv.
        1m      NS      ns.herring.bv.
```

Which values to use in the SOA record is an interesting question. The answer depends on how often your zone will actually change and how important it is for the change to be known immediately. In some settings the zone might be managed as a dynamic zone, but, in fact, the contents will be highly static. Hosts will keep their IP addresses for a long time, and if it does change it is no catastrophe if the change is not known at once. This is true for many office settings. In other settings hosts will change IP numbers fairly often, perhaps disappear, and when they reappear the new IP number should be known ASAP. This could be the case for a dial-up setting. Of course, whether a host changes an IP number often in a dial-up setting might be an administrative decision as well, and it might prove better not to change IP numbers often.

The SOA shown previously is suited for a highly dynamic zone, where hosts change IP numbers often—potentially several times a day. In the opposite case, the SOA values used for the main penguin zone in Chapter 2 are more appropriate. The thinking behind these values is that the serial number will be automatically maintained by the nameserver. It does not keep it in the yyyymmddnn format and giving the impression that it does by seeding the zone with a serial number in that format is liable to cause confusion.

The refresh interval is short. Normally the NOTIFY protocol discussed in Chapter 2 ensures that the update propagates promptly to all slave servers, but the UDP packed bearing the NOTIFY can get lost occasionally. In that case a five-minute refresh interval might not be too onerous to wait out. If the refresh fails, the transfer will be repeated every retry interval—every 2 minutes here. A shorter interval might be used if failures need to be corrected more quickly. This depends on your environment and requirements.

An interesting problem arises if the zone transfer fails repeatedly. How long is it before the zone expires? In a highly dynamic zone, the data will quickly become stale. Having the slave servers fail rather than serve stale data might be preferable, and possibly set off more alarms as well, which could help the situation to be resolved more quickly. Do not keep a zone immediately under a TLD on such a short leash. The TLD administrators can suspend your domain if it tends to be unavailable. For a highly dynamic zone, six hours might be too much.

The one minute TTL was chosen because that way caches will expire the cached data quickly and rerequest the data, which might have changed, often. This helps ensure that new data becomes known quickly. It is important that the minimum TTL be one minute as well because it controls negative caching of the zone. Very few zones will be this dynamic. For most of them much higher intervals are appropriate.

There is also a scaling problem here. BIND will only do whole zone transfers. If your zone is big and changes often, the load on the DNS hosts and the network can grow into a nuisance or even become intolerable. Incremental zone transfers will fix this, but this is not yet available as of June 2000. The problem will be aggravated by having many slaves for the zone. But I do mean big—say, in the order of several thousand hosts.

Note that BIND does some sanity checks on the SOA values: It wants the refresh interval to be at least two times longer than the retry interval, and the expire interval to be at least one week. If these checks are violated, warning messages will be printed in the logs. The messages can be irritating, but if you choose to set the values contrary to the sanity checks it's okay, because you know what you're doing, right? The expire value shown earlier provokes this warning:

```
pz/dyn.penguin.bv: WARNING SOA expire value is less than 7 days (21600)
```

Having edited named.conf and seeded the zone, the nameserver can be "ndc restart"ed, and the zone populated with data. Remember to look in the logs for error and warning messages.

The Client

The DNS update client can either be a DHCP server or a machine that gets a dynamic address assigned to it by some mechanism, such as PPP or DHCP. Which model to use is an administrative policy decision, to which I will return later.

The nsupdate command-line tool is for making updates to dynamic zones. It can be used by hand or scripted. As a standard UNIX tool, it will read commands from `stdin`, print messages to `stdout`, and send errors to `stderr`.

With this tool you can edit zones, unconditionally or conditionally. The point of conditional edits is to enforce administrative policies. If a DHCP server knows that it should never enter new names into the zone, it might supply the condition that the name already exists before doing any updates. On the other hand, if the DHCP server knows that it should never overwrite a name that is already present in the zone, it might submit that as a condition for the update. All updates are atomic: The conditions and updates are submitted as one request, which is either acted upon or rejected because of failed prerequisites.

12 0789722739 CH09 10/10/00 6:45 PM Page 169

Update Prerequisites

There might be no prerequisites, but if there are, these are available:

- The name does not exist within the zone; no record of any type matches the name. Syntax: `prereq nxdomain` *`domain-name`*
- The RRset exists; at least one record exists for the name. Syntax: `prereq yxdomain` *`domain-name`*
- Specified RR exists for the name, no matter what value. Syntax: `prereq nxrset` *`domain-name [class] RR-type`*
- The specified RR exists with the specified value. Syntax: `prereq yxrrset` *`domain-name [class] RR-type RR-data`*

Update Actions

The only possible actions are deleting and adding records:

```
update delete domain-name [class] [RR-type [RR-data...]]
```

The delete operation may remove all records in an RRset, all records of the specified type, or all records with the specified type and value:

```
update add domain-name ttl# [class] RR-type RR-data
```

The add operation requires a nonzero TTL value. The syntax beyond the operation keywords is identical to what you find in a zone file. As in zone files the class is optional, and the class IN is the default.

The specified updates are aggregated into one DNS update request, and it is only performed if all the given prerequisites are met. The update is atomic with regard to other possible update requesters as well, so the operation should always work as expected. One request might only test prerequisites and perform edits of one and the same zone. When performing updates in dyn.penguin.bv, only names within that zone can be tested by the prerequisites.

Some checks are performed on the update data. For example you are not allowed to add a CNAME record to a name that already has records, and likewise, you're not allowed to add anything to a domain name that already has a CNAME record.

Using nsupdate

Knowing all this should allow us to operate nsupdate safely. I'm going to sign the update requests using the key generated earlier in this chapter. This command assumes I left the keys in /etc. The part after the colon (:) specifies the name of the key, not the name of the key file:

```
# nsupdate -k /etc:ns.penguin.bv.
> update add magellan.dyn.penguin.bv 1h A 192.168.55.200
> (blank line)
```

Upon getting the empty line, nsupdate sends an update request to the primary master server. The primary master server will check whether the client is allowed to perform the specified edits and, if so, check the prerequisites. Then, if they are met, it performs the updates all at once. You can now exit nsupdate, press Ctrl+D. Note that pressing Ctrl+D without entering the empty line first will abort the operation. It will not be submitted to the server. You can now test whether the update had any effect:

```
# dig @ns.penguin.bv. magellan.dyn.penguin.bv. ANY +pfmin
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62041
;; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUERY SECTION:
;;      magellan.dyn.penguin.bv, type = ANY, class = IN

;; ANSWER SECTION:
magellan.dyn.penguin.bv.  1H IN A  192.168.55.200
```

The A record I submitted with nsupdate is in place, so it worked. If you're not using TSIG, but rather IP number authentication, the `-k` option should not be used. Now the next job is getting the change to the slave servers.

nsupdate is not very good about displaying errors unless you specify the `-d` (debug) switch. But keeping an eye on the standard messages log file on the server will let you know whether the update was rejected due to authentication problems:

```
unapproved update from [192.168.55.2].2391 for dyn.penguin.bv
```

The contents in the zone log and ixfr files will tell you what updates arrive and are approved. This is the content of the dyn.penguin.bv.ixfr file after adding the record shown previously:

```
;BIND LOG V8
[DYNAMIC_UPDATE] id 27514 from [192.168.55.2].2390 at 958316354
        (named pid 3814):
zone:   origin dyn.penguin.bv class IN serial 9
update: {add} magellan.dyn.penguin.bv. 60 IN A 192.168.55.200
```

Similar information can be found in the zone log file. If you look at the zone file right after the update, it will (probably) not have been updated with the new information. Rewriting large zone files whenever they are updated is resource consuming. Instead the ixfr and log files are appended with the changes done, and upon occasion BIND will roll all the changes into the zone file and rewrite it. The log and ixfr files are transaction logs as used in databases and modern file systems. When an update is applied to the zone, BIND writes the updates to the logs at once. That way, if the server or BIND crashes, BIND can do a roll forward of the transaction log when BIND restarts and no changes will be lost.

The SOA serial number is updated automatically when an update is received. You do not need to update the SOA record.

Slave Server Issues

In the setting where the dynamic zone does not really change that often and immediate propagation of changes is not critical, there is nothing to add to what has already been said about slave servers in Chapter 2.

In the case where the dynamic zone changes often and immediate updates of slave servers is important, I would like to remind you of, and reinforce, some things said in Chapter 2. When an update is received, the server will follow the usual procedure when a zone is updated: After a random waiting period (in the order of tens of seconds), a NOTIFY request is sent out to all the listed nameservers and any additional servers listed in named.conf. In this setting keeping the list of nameservers up to date and ensuring that they really receive NOTIFY requests becomes important. Combining NOTIFY requests with appropriate intervals in the SOA record will ensure that your slave servers are current and up-to-date;.

There is one more thing. As long as your dynamic zone is not right under a TLD (as is the case for dyn.penguin.bv) and is almost exclusively used internally, which will also often be the case, the need for a slave server might not be that great—especially in small places. One server can handle large amounts of requests for the zone, so load will not be an issue. It is more an issue of reliability and redundancy, which might be less important in the dynamic zone, but more important in larger organizations.

Reverse Zones

I have not mentioned reverse zones at all thus far into the chapter. Of course you will need reverse zones as well as forward zones, and they are maintained exactly the same way. In the case of classlessly delegated reverse zones, you should send update requests for the real record name, not the correct name. In the case of the classless emperor.penguin.bv network in Chapter 2, the update request would be for names such as 129.128-255.56.168.192.in-addr.arpa, not 129.56.168.192.in-addr.arpa.

A One Host Zone

In some settings not having to set up a separate dynamic zone for dynamic updates would be the best scenario; or to enable a specific host, or key, to modify only the records of one specific domain name, a more fine-grained access control of who may change what. In the setting shown previously, anyone with the correct key or access to the right host is able to perform any updates on the whole zone. This might not be desirable, and if you find yourself in such a situation, you should consider not implementing dynamic DNS at all. If you can't trust your users at this level, they should perhaps not be able to alter DNS at all.

But there is a "hackish" way to work around it: It is possible to make a "one host" zone. The zone can have its own update ACL and thus the holder of the associated key or IP number can only update the zone, not anything outside it, and no one else can alter the zone either. The way to do this is to delegate the zone bearing the hostname to the nameservers you want, as shown previously, and then seed the zone. If the zone is for magellan.penguin.bv,

```
$TTL 1m
;
@       1m      SOA ns.penguin.bv. hostmaster.penguin.bv. (
                1               ; serial
                5m              ; refresh
                2m              ; retry
                6h              ; expire
                1m              ; minimum
                )
        1m      NS      ns.penguin.bv.
        1m      NS      ns.herring.bv.
        1m      A       10.10.10.10
```

it gives magellan.penguin.bv an A record with the value 10.10.10.10. This can be deleted and re-added just as described previously for magellan.dyn.penguin.bv. This gives finer update access control and the capability to have dynamic hosts directly under the main domain, but at the cost of configuration overhead and increased key/ACL maintenance. Of course, anyone able to update this zone can add subdomains of magellan.penguin.bv if he wants to, so it can't really be called secure or considered very restricted.

DHCP

DDNS and DHCP are complementary services. DHCP doles out the addresses and DNS helps you find the address.

The ISC is also implementing a DHCP server for UNIX systems. As I write this, DHCP 2.0 is the production version and in common use. DHCP 3.0 is in beta. The 2.0 distribution does not support dynamic DNS updates. The 3.0 beta does support dynamic updates but the documentation carries big warnings about not being final, so use it at your own peril and only if you need the features. But a production release might be available by the time you read this. As with BIND, you can get DHCP from the ISC ftp site: `ftp://ftp.isc.org/isc/dhcp/`. See its Web site at `http://www.isc.org/products/DHCP/` for more information about the available releases, their status, and features. I will not even try to describe the design, implementation, and usage issues connected with DHCP; I will simply discuss some DHCP/DNS integration issues.

Please see *The DHCP Handbook* for more complete information about DHCP, both the standard and the implementation.

Mixing DNS and DHCP Implementations

Some people want to use Windows DHCP with BIND DNS, or vice versa. I have not had the opportunity to try either combination, but the general advice available on the Net about this is "don't." It apparently works better if you keep Windows DHCP paired with Windows DNS and ISC DHCP paired with ISC DNS.

DHCP and Static DNS Entries

Due to the lack of support for dynamic DNS support in DHCP 2 and, more significantly perhaps, the potential management overhead if everyone could grab any name and get an IP address to go, a lot of sites use fixed names for their DHCP range. In BIND 8 it's quite easy to enter such ranges in zone files too, using $GENERATE which was introduced in Chapter 2. In such a case, the 16 new penguin employee computers would be assigned an IP range, such as 192.168.55.220 to 229, and the names would be entered thus:

```
$GENERATE 220-229 dhcp$ A 192.168.55.$
```

The names would be dhcp220.penguin.bv and so forth. This is a good way to do it; it is simple and low maintenance. For hosts that you want to have fixed IP numbers or fixed hostname, ISC DHCP 2 provides a way for you to assign them. In the dhcpd.conf file, insert something like this:

```
host gentoo {
        hardware ethernet 00:60:1d:1f:1e:f7;
        fixed-address 192.168.55.55;
}
```

This assigns the given IP address to the host bearing the given Ethernet address. Just enter the name in DNS in the usual manner. The "gentoo" part of the host statement is arbitrary, but it would be good policy to assign 192.168.55.55 the name gentoo.penguin.bv.

DHCP and Dynamic DNS Entries

As I mentioned earlier, version 3 of the ISC DHCP distribution can do dynamic updates of DNS based on the hostname the client wants. However, the how of this integration has not been entirely worked out at this time so I'll refrain from teasing you with what you can't do. *The DHCP Handbook* has more information about how it is supposed to work.

The DNS update conditionals allow the DHCP server to specify update conditions to the DNS server such as "if the name already is in use" to forbid users from using new names or "if the name is not present" to forbid users from using names already in use. Whichever way you want it is a pure administrative decision and what you can allow depends on how much you trust your users. If you don't trust your users, I recommend that you give them static names.

Dynamic Updates by the Client

I have assumed that the DHCP server would do the DNS updates. Of course it does not have to, and indeed, on a limited scale it might be easier to do on the client's server. Doing it from the DHCP server gives low ACL/key maintenance overhead and, if need be, full control of what gets added. But there is nothing stopping you from giving the DHCP (or PPP) client access to update DNS. By running a simple script after the interface has been assigned, an address DNS can be updated:

```
#!/bin/sh

PATH=/sbin:/usr/sbin:/bin:/usr/bin
export PATH

IF=hme0
NAME=gentoo.dyn.penguin.bv
TTL=60

IP=`ifconfig $IF | awk '/inet/ { print $2; }'`

nsupdate <<EOC
update delete $NAME
update add $NAME $TTL A $IP

EOC
```

The script shown here works on Solaris. It needs to be adapted to other OSes; the interface names and the output of ifconfig vary wildly between OSes. Also add the -k option if you want to use TSIG signing.