BIND, dynamic DNS

From FreeBSDwiki

Contents

The task

You've got your own BIND server with a static, public IP address, and your own domain which you host on it. You've also got one or more machines on dynamic public IP addresses - perhaps your or your customers' or friends' home machines, or small offices in areas that don't offer static addresses - and you want to use your own equipment to maintain DNS records to point to the machines on dynamic addresses, rather than using third-party solutions.

Checking versions of BIND and its tools

In order to set up dynamic DNS on your server, first you need to make sure you're running BIND9 or better - as of this article, you want BIND 9.3.1.

```
server# which named
/usr/sbin/named
server# named -v
BIND 9.3.1
```

```
client# which named
/usr/sbin/named
client# named -v
BIND 9.3.1
```

Okay, good. But we also need to dig a little further, because FreeBSD systems have a nasty habit of shipping with some elderly BIND8 components higher up in the PATH than the newer BIND9 versions that go with the actual server. Specifically, we need to make sure we're using the new version of nsupdate, which we'll be using to do the dynamic updates from client to server:

```
client# where nsupdate
/usr/sbin/nsupdate
/usr/bin/nsupdate
```

Aha - there are two copies of nsupdate on this machine! Now we need to see which one of them is higher up in the PATH (and therefore will be the one that runs if you don't specify which one you want), and whether they're both the same version or not:

```
client# which nsupdate
/usr/sbin/nsupdate
client# ls -l /usr/bin/nsupdate && ls -l /usr/sbin/nsupdate
-r-xr-xr-x  1 root  wheel  1252248 May  8  2005 /usr/bin/nsupdate
-r-xr-xr-x  1 root  wheel   245324 Jul  5  2004 /usr/sbin/nsupdate
```

AHA! As we suspected, there's a copy of the **nsupdate** from BIND8 lurking in our PATH higher up than the BIND9 version - and BIND8's **nsupdate** tool was completely broken and useless. So, we'll get rid of it. (Obviously, if you don't have an older version in the way, you don't need to do this step - but it's important to check and make sure, because you'll be tearing your hair out later wondering why everything *looks* like it's working but *isn't* if you have this problem but don't catch it.)

```
client# rm /usr/sbin/nsupdate
```

With that taken care of, we can start working on the subdomain we want to dynamically update. In this example, we're going to use a (fictitious) parent zone, server.net, which is maintained by a statically-addressed FreeBSD server which we have (root) control of, and we already have functional DNS for the parent zone.

Preparing a "seed" zone file

First, we need to prepare a "seed" zone file for the subdomain we want to be able to dynamically update. In this example, our dynamic subdomain is going to be **client.server.net**. This zone file should be very minimal - we only want to put the barest amount of information in here, to define those parts of the domain that WON'T ever change. In this case, that will be the SOA record, the NS records, and the MX record. (Since MX records are based on A records, not on IP addresses, the MX *record* won't change even when the IP address of the mailserver itself does).

```
$ORIGIN .
$TTL 10 ; 10 seconds
client.server.net   IN SOA  ns1.server.net. hostmaster.server.net. (
                            18        ; serial
                            10800     ; refresh (3 hours)
                            3600      ; retry (1 hour)
                            604800    ; expire (1 week)
                            10        ; minimum (10 seconds)
                            )
$TTL 3600      ; 1 hour
                    NS     ns1.server.net.
                    NS     ns2.server.net.
                    MX     10 client.server.net.

$ORIGIN client.server.net.
```

Generating a cryptographic key

While it's possible to allow zone updates without any cryptographic security, it's certainly not recommended - and implementing the crypto isn't difficult, anyway, so let's get to it. We're storing our zones in **/etc/namedb/zones**, and we'll park our key(s) in **/etc/namedb/zones/keys**.

```
server# mkdir /etc/namedb/zones/keys
server# cd /etc/namedb/zones/keys
server# dnssec-keygen -b 512 -a HMAC-MD5 -v 2 -n HOST client.server.net.
Kclient.server.net.+157+15661
server# ls -l
-rw-------  1 root  wheel  134 May 20 19:46 Kclient.server.net.+157+15661.key
-rw-------  1 root  wheel  145 May 20 19:46 Kclient.server.net.+157+15661.private
```

And there they are - one public key, one private key. The next step is incorporating them into the named.conf file.

Setting up named.conf

First, we need to pluck the actual value of the private key out of its file to insert it directly into the zone definition.

```
server# cat /etc/namedb/zones/keys/Kclient.server.net.+157+15661.private
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: omr5O5so/tZB5XeGuBBf42rrRJRQZB8I9f+uIIxxei8qm7AVgNBprxtcU+FQMzBvU/Y+nyM2xbs/C8kF3eJQUA==
```

That last bit of the private key is what we need. So, we copy and paste it into the new zone definition and key reference we're appending to **/etc/namedb/named.conf**:

```
key client.server.net. {
    algorithm "HMAC-MD5";
    secret "omr5O5so/tZB5XeGuBBf42rrRJRQZB8I9f+uIIxxei8qm7AVgNBprxtcU+FQMzBvU/Y+nyM2xbs/C8kF3eJQUA==";
};

zone "client.server.net" {
    type master;
    file "zones/client.server.net";
    allow-update{
        key client.server.net;
    };
};
```

Now that we have the keys set up, we need to make sure nobody can read them, either in their original directory *or* in the line we just added to **named.conf** with the value of the private key:

```
server# chmod -R 400 /etc/namedb/zones/keys;
server# chmod -R 400 /etc/namedb/named.conf;
```

And we're done. If you like, you may also **chmod 400 /etc/namedb/zones**, but it's not strictly necessary since everything in there is available by normal DNS query from the internet anyway. The only thing left to do on the server side is restart named and make sure it still works!

Restarting and testing BIND at the server

```
#server ps ax | grep named
76949  ??  Ss     0:01.03 named
#server kill 76949
#server named
#server ps ax | grep named
81230  ??  Ss     0:00.49 named
```

Ok, we've found and killed our previous instance of BIND (don't just use -HUP - you need to kill it all the way), then gotten it back up and confirmed it's running. Now let's see if it responds properly when we ask it about the new zone:

```
#server dig @localhost client.server.net
; <<>> DiG 9.3.1 <<>> @localhost ANY client.server.net
; (1 server found)
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13783
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;client.server.net.          IN     ANY

;; ANSWER SECTION:
client.server.net.     10     IN     SOA    ns1.server.net. hostmaster.server.net. 18 10800 3600 604800 10
client.server.net.     3600   IN     NS     ns1.server.net.
client.server.net.     3600   IN     NS     ns2.server.net.
client.server.net.     3600   IN     MX     5 client.server.net.

;; ADDITIONAL SECTION:
ns1.server.net.        21600  IN     A      63.126.69.120

;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sat May 20 20:15:36 2006
;; MSG SIZE  rcvd: 231
```

Excellent - that's exactly what we were looking for; we don't have an A record for client.server.net yet, but the SOA, NS, and MX records - all of which are static - are up and responding perfectly. If you aren't so lucky and get a SERVFAIL or other unfriendly response, try a **tail -n 500 /var/log/messages | grep named** and see if you can narrow your problem down. Once you've got the server responding properly, grab copies of both the .private and the .key file, and get them to your dynamic client box.

Updating the zone from the client

What I did was make a user account specifically for the purpose of handling my automatic updates - since there's no need for any special privileges on the client's end, it's the best way to keep things nice, tidy, and non-risky.

```
client# pw useradd ddns -s /sbin/nologin -d /usr/home/ddns
client# mkdir /usr/home/ddns
client# chown ddns /usr/home/ddns
```

Now let's park our keys in there, however you decided to move them. It's usually best with files this small to just copy and paste them from one window to another, rather than putting copies in less-secure folders that you can scp from and possibly forgetting you left them there - but whatever works for you is fine, as long as they end up in **/home/ddns** on **client** and you get rid of or secure any extra copies you made elsewhere.

```
client# cd /usr/home/ddns
client# ls -l K*
-rw-------  1 root  wheel  134 May 20 19:46 Kclient.server.net.+157+15661.key
-rw-------  1 root  wheel  145 May 20 19:46 Kclient.server.net.+157+15661.private
```

And there they are. Note: even though we only specifically refer to the .private key in the following script, nsupdate **does** require the presence of the .key file in the same directory. ALSO note: the keys MUST be in the working directory **nsupdate** is called from; there is a bug in the version of nsupdate currently shipping (BIND 9.3.1) which causes it to fail if it tries to read keys outside of its own directory!

Before we get started with our client-side scripts, please note that in addition to having BIND 9.3.1 or better (and matching versions of the BIND tools such as **nsupdate**) installed, your client system will also need **/usr/ports/www/p5-libwww** so that it can fetch HTML pages containing its effective public IP address.

Once that's squared away, we'll proceed to the customized stuff:

set-ddns.pl (a freebsdwiki.net original)

This script does two things for you: first, it parses your WAN IP address out of a web page. **If at all possible, I strongly recommend that you tailor the regex and the URL used to fetch your WAN IP from your router instead of using a remote server.** You can also check the set-ddns.pl router settings list to see if somebody's already done the work for you - or to add details for your own router, if you have a model nobody's covered yet and end up sussing it out yourself!

If you can't figure out how to parse the WAN address out of your router, you can set up a simple web page using php. In the default example, we'll assume that's what you're doing, and that you're hosting this page at http://server.net/ip.php. (Note: THIS METHOD REQUIRES A WORKING PHP INSTALLATION!)

```
<html>
<head>
<?php $ip = $_SERVER['REMOTE_ADDR']; ?>
<title><?php echo $ip; ?></title>
</head>
<body>
<?php echo "Current IP Address: " . $ip; ?>
</body>
</html>
```

If all else fails, http://checkip.dyndns.org will work in the place of a self-hosted copy of the php script shown above. But I don't recommend it. For one thing, you're relying unnecessarily on somebody else's stuff; for another thing, you're unnecessarily taxing somebody else's resources - and the whole point is to get this done on your own hardware! Anyway, get it working whichever way you choose, and then we're ready to move on to the fun stuff.

Once **set-ddns.pl** has parsed your WAN IP address (or addresses, if you have a multi-homed network - see the Xincom Twin-WAN Router section in the router config list; fun stuff!) out of whatever you've given it to work with, it then calls on nsupdate to update your zone file with the data.

Without further ado, here it is! As you're entering / once you've entered in the script, be mindful of its user-configurable variables: in general, the UPPERCASE variables are things that you should set yourself, while lower or mixed-case variables are used internally and you probably won't need to mess with.

```
#!/usr/bin/perl

# set-ddns.pl
#
# Copyright (c) 05-20-2006, JRS System Solutions
# All rights reserved under standard BSD license
# details: http://www.opensource.org/licenses/bsd-license.php

# note: requires BIND 9.3.1 and CPAN LWP::UserAgent, HTTP::Request, and HTTP::Response
#       FreeBSD admins may find the CPAN modules under /usr/ports/www/p5-libwww
#
# WARNING: FreeBSD admins must make CERTAIN they are calling the BIND9 version
#         of nsupdate - FreeBSD systems have a nasty habit of leaving a copy
#         of the BIND8 version higher up in the PATH, even in systems shipped
#         with BIND9 in the base install!
#
# Updated to adhere to Modern Perl conventions (see
# http://perl-begin.org/tutorials/bad-elements/ ) by Shlomi Fish
# ( http://www.shlomifish.org/ ).
# Copyright (c) 15-April-2012, Shlomi Fish
# Under standard BSD license
# details: http://www.opensource.org/licenses/bsd-license.php

use strict;
use warnings;

use LWP::UserAgent;
use HTTP::Request;
use HTTP::Response;

my $NAMESERVER = 'server.net.';
my $KEYDIR = '/usr/home/ddns';
my $KEYFILE = 'Kclient.server.net.+157+15661.private';
my $TYPE = 'A';
my $TTL = '10';
my $HOST = 'client.server.net';

my $url_string = 'http://server.net/ip.php';
my $ua = LWP::UserAgent->new;
my $req = HTTP::Request->new('GET',$url_string);
my $resp = $ua->request($req)->as_string();

my $WAN;
```

```
if (not
   (
   ($WAN) =
   $resp =~ m/Current IP Address\: (\d{0,3}\.\d{0,3}\.\d{0,3}\.\d{0,3}).*/s
   )
   )
{
   die "Cannot match Current IP Address in response.";
}

print "==============================\nWAN IP parsed: $WAN\n==============================\n";

chdir ($KEYDIR);

open (my $nsupdate_fh, "| /usr/sbin/nsupdate -k $KEYFILE")
   or die "Cannot open nsupdate! $!";
print {$nsupdate_fh} "server $NAMESERVER\n";
print {$nsupdate_fh} "update delete $HOST A\n";
print {$nsupdate_fh} "update add $HOST $TTL A $WAN\n";
print {$nsupdate_fh} "show\n";
print {$nsupdate_fh} "send\n";
close ($nsupdate_fh) or die "Cannot close nsupdate - $!.";
```

OK, that should be it - let's lock down our permissions:

```
client# chown -R ddns /usr/home/ddns
client# chmod 400 /usr/home/ddns/Kclient.server.net*
client# chmod 500 /usr/home/ddns/set-ddns.pl
```

Done - now only the neutered, no-shell-allowed user "ddns" (or root, of course) can access our keys or our script. So let's fire it up and see if everything's working right!

```
client# /usr/home/ddns/set-ddns.pl
==============================
WAN IP parsed: 24.38.194.62
==============================
Outgoing update query:
;; ->>HEADER<<- opcode: UPDATE, status: NOERROR, id:     0
;; flags: ; ZONE: 0, PREREQ: 0, UPDATE: 0, ADDITIONAL: 0
;; UPDATE SECTION:
client.server.net.  0      ANY     A
client.server.net.  10     IN      A      24.38.194.62
```

Outstanding! If your results didn't come out right the first time, check to see if your WAN IP parsed correctly, then check the output of **nsupdate** below it for ZONE FAILED, UPDATE REFUSED, or ROLLFORWARD FAIL messages. If you got a ROLLFORWARD FAIL message, that basically means that you messed with the seed file on the server after you'd begun dynamically updating it - but don't let that scare you, you'll just have to go rm its journal file. Journal files are typically found in the /var working mirror of namedb; in the example above it would be at /var/named/etc/namedb/zones/client.server.net.jnl.

If your server does update, but you get "; Communication with server failed: timed out" on the client, add "-v" to the open call: *open (NSUPDATE, "| /usr/sbin /nsupdate -k $KEYFILE -v");*

Setting up a crontab

Once you get your first successful update, all that's left to do is set up a crontab on your client box to keep the updates happening automagically. Fire off a **crontab -u ddns -e** at the prompt (possibly doing a **setenv editor ee** first, if you're allergic to vi) and add this tab (which should be the ONLY tab on our new and neutered **ddns** user):

```
* * * * * /usr/bin/perl /usr/home/ddns/set-ddns.pl
```

And that should be it! From here on out, cron will call your new tab once every 60 seconds, and issue updates to any records you've set to update. If you like, you can even get fancy and add checks to the script to keep it from issuing unnecessary updates when the IP hasn't changed. You might also consider setting up fancy tricks with TCP redirectors or VPN tunnels like datapipe or OpenVPN to follow you through your IP changes; or if you have a multi-homed network you might want to set up a "failover" A record to automatically send you to the lower latency of two IPs for the same machine.

If you come up with any particularly interesting bells, whistles, or parlor tricks not covered here, be sure to let us know!

For setting up "failover" A records when you have more than one IP address resolving to a particular server, see also: BIND, dynamic DNS, failover A records.

To get your WAN IP address, see set-ddns.pl router settings list.

See Also

The DynDNS article contains instructions on how to configure public internet Dynamic-DNS service providers.

Samba has an implementation of Microsoft's Windows Internet Name Server (WINS) service which offers dynamic registration for and resolution of NetBIOS names.

Categories: Common Tasks | FreeBSD for Servers | DNS
