

Installing djbdns for Name Service Part Two

Jeremy Rauch 2000-07-10

1. Introduction

Traditionally, BIND has been the nameserver of choice when doing name service on a Unix system. Like many of its close relatives, such as sendmail, it was designed at a time when the internet wasn't even known as the internet, and security wasn't a concern. This has caused more than a few problems over the years, and many point to the age of its codebase, and lack of designed-in security as part of the problem.

djbdns, formerly known collectively as DNSCache, is a replacement for BIND, written by Dan Bernstein, the author of qmail, a mail transport agent that has been rapidly growing in popularity in the last few years. djbdns, like qmail, was written as a replacement to the traditional program used for its service, keeping the needs and requirements of the modern internet in mind. By taking a more modular approach, it has become possible to reduce code sizes in to smaller, more manageable chunks, reduce the number of lines of codes run with privileges, and take a more practical approach to security.

In this, the second part of our series on djbdns, we'll discuss some techniques for setting up a secondary nameserver. There are a few different methods we can use, and we'll discuss the merits of each. Finally, we'll set up *walldns* for doing reverse resolution on IP's for the external world, as a demonstration of some of the different-from-conventional thinking being employed in djbdns to improve security. When we're done with this article, you should have all the building blocks to set up djbdns as a true replacement to BIND.

Before reading this article, please make sure you've read [the first article](#) so you understand how things are laid out.

2. Setup and configuration of *axfrdns* to allow for a traditional secondary DNS

There are a number of different ways to approach setting up secondary DNS when it comes to djbdns. Under traditional name servers, you have a primary and a secondary (or multiple secondary) nameservers. The primary nameserver is where the zone files are edited. The secondary nameservers will perform a zone transfer. This is done over TCP port 53. Essentially, the secondary nameserver will pull all the records for the zone it is acting as a secondary for. This method tends to be fairly slow, and due to the monolithic nature of BIND, some find it can be difficult to test and work with.

djbdns can support this zone transfer over TCP using the *axfrdns* daemon. *axfrdns* uses the *ucspi-tcp* suite of tools

to allow zone transfers to work properly. It's relatively easy to install and setup.

The following steps will install axfrdns. For this example, we have a copy of tinydns running on the IP 10.20.30.13. tinydns uses UDP port 53, and axfrdns uses TCP port 53, so they can peacefully coexist.

- o **Download and install ucspi-tcp**

<http://cr.yip.to/ucspi-tcp/ucspi-tcp-0.88.tar.gz>

ucspi-tcp is a suite of tools that can be used to build client server applications. *axfrdns* will use it to listen and respond to TCP dns requests.

- o **Compile and install ucspi-tcp**

As will all tools written by Dan Bernstein, ucspi is easy to compile and install

```
> gunzip -c ucspi-tcp-0.88.tar.gz | tar xf -
```

Will decompress and detar ucspi-tcp. The source will be placed in a directory named ucspi-tcp-0.88/.

```
> cd ucspi-tcp-0.88
```

We will install everything in /usr/local. If this isn't to your liking, simply edit the conf-home file. If you don't use gcc, edit conf-cc, and if you don't use gcc as a linker, edit conf-ld. Pretty simple.

To compile, simply run:

```
> make .
```

The build environment is designed to determine what settings are needed without going through the often slow, often over complex gyrations that something like autoconf goes through.

Once the compilation completes, run:

```
> su
Password:
# make setup check
```

This will install the ucspi-tcp programs in /usr/local/bin.

- o **Configure and set up axfrdns**

Like all of the tools in the djbdns suite, axfrdns comes with a simple program to set it up for you. This program is *axfrdns-conf*. Setting up axfrdns is as simple as running this command with the proper arguments and editing a simple configuration file that limits the hosts that can perform a transfer. First, we'll need to create a few accounts.

```
# /usr/sbin/useradd -d /var/dnscache -s /bin/false -c axfrdns axfrdns
# /usr/sbin/useradd -d /var/dnscache -s /bin/false -c axfrlog axfrlog
```

This will create the accounts axfrdns will run under.

Finally, set up axfrdns. It's going to be listening on IP 10.20.30.13, just like the tinydns process on this machine.

```
# /usr/local/bin/axfrdns-conf axfrdns axfrlog /var/dnscache/axfrdns /var/dnscache/
tinydns 10.20.30.13
```

This will install axfrdns in */var/dnscache/axfrdns*, running as axfrdns, logging as axfrlog, on ip 10.20.30.13. It knows the tinydns process on this machine lives in */var/dnscache/tinydns*, and knows to look in */var/dnscache/tinydns/root* for its data files.

We take a final step in installing axfrdns before letting *svscan* begin to manage it. We need to configure ACL's to prevent unauthorized sites from connecting. You'll need to use your favorite editor to modify the file in question. We'll simply cat it, so you can see what should be in it.

```
> cat /var/dnscache/axfrdns/tcp
# sample line: 1.2.3.4:allow,AXFR="heaven.af.mil/3.2.1.in-addr.arpa"
10.20.30.14:allow,AXFR="internal/204.200.10.in-addr.arpa"
:deny
>
```

This will allow the host 10.20.30.14 to perform zone transfers for the internal domain, as well as its reverse, 204.200.10.in-addr.arpa. All other clients are explicitly denied.

Now we can let svscan know about axfrdns, and get things running

```
# ln -sf /var/dnscache/axfrdns /service
```

o Testing

From 10.20.30.14, its easy to test whether or not we can perform zone transfers. This assumes 10.20.30.14 has the djbdns package installed.

```
# cd /tmp
# /usr/local/bin/tcpclient 10.20.30.13 53 axfr-get internal data data.tmp
# cat data
#963128650 auto axfr-get
Zinternal:a.ns.internal.:hostmaster.internal.:963128650:16384:2048:1048576:2560:2560
&internal::a.ns.internal.:259200
+a.ns.internal:10.20.30.13:259200
&internal::b.ns.internal.:259200
+b.ns.internal:10.20.30.14:259200
@internal::a.mx.internal.:0:86400
+a.mx.internal:10.20.30.4:86400
+gate.internal:10.20.30.1:86400
+lp.internal:10.20.30.2:86400
+hotate.internal:10.20.30.3:86400
+mail.internal:10.20.30.4:86400
+ebi.internal:10.20.30.5:86400
+uni.internal:10.20.30.6:86400
+unagi.internal:10.20.30.10:86400
+ns.internal:10.20.30.13:86400
+ns2.internal:10.20.30.14:86400
+hamachi.internal:10.20.30.12:86400
#
```

Looks good. If your secondary nameserver is running BIND, you should be able to configure it as you normally would. If you're going to use tinydns as a secondary name server (and you should!), then there are a few options available.

3. Setting up tinydns as a secondary nameserver

Setting up tinydns as a secondary nameserver is easy and straightforward. The difficult part is deciding how to propagate the zone to the secondary. As we showed above, we can use axfr-get to perform a traditional TCP

zone transfer. Alternately, if we have access to both the primary and secondary nameservers, we can make updating the secondary a push operation; every time we alter the data file containing the zone, we simply push the new data.cdb file over to the secondary. If we don't have control of the secondary, there is another option. We can use something like publicfile to make the data.cdb or data file always accessible to the secondary, which can check it at some regular period. We'll discuss setting up all 3.

- **Setting up tinydns to be ready to be a secondary name server**

Setting this up is actually quite simple. First, we set up a tinydns server exactly how we discussed in the first article. This secondary IP is 10.20.30.14. svscan is running on this machine, as per the first article. No dnscache is installed.

```
# /usr/sbin/useradd -d /var/dnscache -s /bin/false -c tinydns tinydns
# /usr/sbin/useradd -d /var/dnscache -s /bin/false -c tinylog tinylog
# mkdir /var/dnscache
# /usr/local/bin/tinydns-conf tinydns tinylog /var/dnscache/tinydns2 10.20.30.14
```

This will create a tinydns that is set to run on 10.20.30.14. We can start the process now -- we will not be configuring this tinydns as we did the previous copy we set up. It will obtain all its zone information from the primary. Please note, the directory name 'tinydns2' is simply to make it clear that this is a secondary name server. You can use tinydns instead if you like.

```
# ln -sf /var/dnscache/tinydns2 /service
```

Now, we simply need to figure out what method for transferring zone information we want to use.

- **Setting up zone transfers using axfr-get**

Setting up a secondary nameserver using axfr-get is pretty simple. We're simply going to write a simple script to grab the zone via axfr-get, and then compile the file into the format tinydns uses (cdb).

```
# cd /var/dnscache/tinydns2/root
# cat > update
#!/bin/sh
cd /var/dnscache/tinydns2/root
/usr/local/bin/tcpclient -i 10.20.30.14 10.20.30.13 53 /usr/local/bin/axfr-get internal data
data.tmp
```

```
make
^D
# chmod 700 update
# ./update
/usr/local/bin/tinydns-data
# ls -l data*
-rw-r--r-- 1 root root 659 Jul 10 00:51 data
-rw-r--r-- 1 root root 3100 Jul 10 00:51 data.cdb
#
```

The secondary on 10.20.30.14 should now answer queries for the internal zone.

```
# dnsq A ebi.internal 10.20.30.14
1 ebi.internal:
113 bytes, 1+1+2+2 records, response, authoritative, noerror
query: 1 ebi.internal
answer: ebi.internal 86400 A 10.20.30.5
authority: internal 259200 NS a.ns.internal
authority: internal 259200 NS b.ns.internal
additional: a.ns.internal 259200 A 10.20.30.13
additional: b.ns.internal 259200 A 10.20.30.14
#
```

The update script should be set up as a cron job.

```
# cat /var/spool/cron/root
0,30 * * * * /var/dnscache/tinydns2/root/update > /dev/null 2>&1
#
```

Using regular zone transfers has some advantages. It's fairly easy to setup, and doesn't require anything complex from the primary. The primary can as easily be a djbdns tinydns server as a BIND server. However, it tends to be somewhat slow. Also, tinydns won't accept (or send) notification when an update needs to take place. This can be problematic when a zone needs to be updated frequently. It also requires the installation of axfrdns on the primary. If the secondary is running BIND, this is really your only option. If its running tinydns, one of the following two options are a better choice.

- o **Transferring zones by pushing from the primary**

When you have control over both the primary and secondary nameservers, and they're both running tinydns, this is really the best method, and its easy too. By copying the data.cdb file directly over, the two nameservers will always be synchronized, axfrdns doesn't need to be run, and the actual transfer can be done securely. In addition, it reduces the overhead of transferring a large zone too often -- the transfer will only take place when the zone has been altered.

To do this, simply edit the Makefile in the primary tinydns' root directory, and modify it to automatically copy the file over to the secondary. This idea is one mentioned in the djbdns FAQ.

```
# cd /var/dnscache/tinydns/root
# cat Makefile
remote: data.cdb
    scp data.cdb 10.20.30.14:/var/dnscache/tinydns2/root
data.cdb: data
    /usr/local/bin/tinydns-data
#
```

Upon running *make* in the */var/dnscache/tinydns* directory on the primary server (you do this to generate the data.cdb file, in case you forgot), it will copy it via ssh to the secondary. If you aren't comfortable scp'ing as root, you can always use the -l option to scp as a different user. Just make sure the user you choose can write the data.cdb to its destination, and that the tinydns process running there can read it.

When you have access to the secondary nameserver, this method is definitely the best. If you don't have access, but it is running tinydns, there is another method we can use.

- o **Transferring zones via publicfile**

One method that works quite well for setting up secondaries when you have no access to the machine is to make either the data or data.cdb file accessible to the secondary host via some other method, be it ftp or http. We'll set up *publicfile*, another set of programs by Dan Bernstein, and use it to publish the data and data.cdb files. The secondary can then use wget, or a similar program, to pull the appropriate file when its been altered.

1. **Download, compile and install publicfile**

At this point, you should be very familiar with compiling and installing things written by Dan. They're all

very consistent.

<http://cr.yip.to/publicfile/publicfile-0.52.tar.gz>

Download publicfile, gunzip and detar it, and compile it exactly as we've done. Follow the directions on <http://cr.yip.to/publicfile/install.html> and you should be fine.

2. Configure and ACL publicfile

Configure publicfile how it tells you to in the installation instructions. We'll place our copy in /var/publicfile.

```
# /usr/sbin/useradd -M -d /var/publicfile -s /bin/false -c publicfile publicfile
# /usr/sbin/useradd -M -d /var/publicfile -s /bin/false -c publiclog publiclog
# ./configure publicfile publiclog /var/publicfile ns.internal 10.20.30.13
```

Next, we need to modify the way the webserver runs, so that only a specific host can connect. Again, use your favorite editor to alter the files; the changed files are cat'd so you can see the changes.

```
# cd /var/publicfile/httpd
# cat run
#!/bin/sh
exec 2>&1

exec envuidgid publicfile softlimit -o20 -d50000 tcpserver -x tcp.cdb -vDRH10 -b50 -c100 0 80 /
usr/local/publicfile/bin/httpd /var/publicfile/file
# cat tcp
10.20.30.14:allow
:deny
# cat Makefile
tcp.cdb: tcp
        tcprules tcp.cdb tcp.tmp < tcp
# make
tcprules tcp.cdb tcp.tmp < tcp
# ln -sf /var/publicfile/httpd /service
```

We then modify the Makefile in /var/dnscache/tinydns/root to copy the data.cdb to a location accessible by the secondary.

```
# cd /var/dnscache/tinydns/root
# cat Makefile
accessible: data.cdb
    cp data.cdb /var/publicfile/file/0/
    chmod 644 /var/publicfile/file/0/data.cdb
data.cdb: data
    /usr/local/bin/tinydns-data
#
```

3. Pull from the secondary

Now, the secondary can simply use `wget` with the `--timestamp` option to pull the `data.cdb` file on some regular basis. If it hasn't changed, the file won't be transferred.

```
> wget -nc -nv --timestamping http://10.20.30.13/data.cdb 03:25:07 URL:http://10.20.30.13:80/
data.cdb [510/510] -> "data.cdb" [1] >
```

This can also be placed in a cron job, and run periodically, similar to how we set up `axfr-get` in the 'update' script. If the file hasn't changed, the `--timestamping` option will prevent it from being repulled. `Wget` is available at [ftp://ftp.gnu.org/pub/gnu/wget](http://ftp.gnu.org/pub/gnu/wget).

. Notes:

If you're running `tinydns` behind a `dnscache`, like we did in the last article for our internal nameservers, and you want to set up secondary, you'll notice quickly that it won't work using `axfrdns` out of the box. `dnscache` listens on both TCP and UDP port 53. The best solution, since you're likely to control both the internal primary and secondary, is to copy the `data.cdb` file, as outlined above, in method 2 for transferring zones. If you really want to use `axfrdns`, the easiest way is to bind it to a virtual ip. Read the manpage for `ifconfig` on your system for information on how to create a virtual.

Using `ftp` under `publicfile` instead of `http` for transferring data files won't work well, as the `wget --timestamping` option won't work properly.

`BIND` has the ability to notify a secondary when the zone has been altered, and trigger a zone-transfer. `djbdns` does not.

All zone information for tinydns is contained in one file, data, which is compiled in to data.cdb. Setting up each zone in a separate file, and modifying the Makefile for data.cdb to cat them all together is probably the cleanest way to manage the domains. To support secondary servers without using axfr-dns, you'll need to transfer the individual zone files, and cause the secondary to recompile the database. The database format used is designed to be very efficient to generate, so this isn't as terrible as it sounds.

For more in depth information on tinydns, dnscache, and the whole djbdns suite, consult <http://cr.yp.to/djbdns.html>

• Conclusion

Hopefully, you've come to the conclusion that djbdns isn't all that difficult to install or use. In some ways, its more straightforward than BIND. As it was designed from the get go with security in mind, you can be sure it won't suffer from some of the more common problems that afflict other DNS software. If you're looking for a secure alternative to what is out there for DNS software, you really need look no further.

With the ability to set up secondary DNS, and interact well with BIND, there's no reason to need to run BIND in most situations. One of the things people seem to have the most trouble with is properly configuring secondary DNS. Using tinydns and the rest of djbdns makes this easy to do.

Relevant Links

Subscribe to the [FOCUS-Linux Mail List](#)

by Security Focus Inc.

[Pluggable Authentication Modules, Part I](#)

by Jeremy Rauch

[Pluggable Authentication Modules, Part II](#)

by Jeremy Rauch

[Libnet 101](#)

by Mike Schiffman

[Installing djbdns \(DNSCache\) for Secure Nameservice](#)

by Jeremy Rauch

[Installing djbdns \(DNSCache\) for Secure Nameservice, Pt. 2](#)

by Jeremy Rauch

[Basic File Integrity Checking](#)

by Jeremy Rauch

[djbdns Homepage](#)

Dan Bernstein

[Privacy Statement](#)

Copyright 2006, SecurityFocus