

DNS for a Wireless Network

Michael J. Martin

Running your own DNS implementation is a requirement for the WLAN solution. First, you need to provide some way for users to resolve domain name requests. Since only http/https traffic is permitted out of the WLAN environment, the DNS server needs to be locally accessible. If you really don't want to run a DNS server or already have one that you use for your domain (that you don't mind WLAN users accessing) you can use reverse NAT on the IOS router to provide local IP-level access to your server. In addition, you need to configure some basic host entries for the proxy server in order for proxy auto-discovery to work. If you have a local DNS server already, you can just add the domains and host entries and skip the DNS server build.

A DNS server consists of three parts:

1. The Unix/Linux daemon called *named*. *Named* is one of the many DNS related binaries distributed as of the Berkeley Internet Name Domain (BIND) distribution maintained by the Internet Systems Consortium (ISC). The BIND distribution is the "standard" DNS distribution used by most Linux implementations, Sun Solaris and Apple's OS X. The current release is 9.3.2. It is a good idea to keep your BIND distribution up to date. DNS is a vital network service and vulnerabilities are exploited almost as quickly as they are patched. In addition to *named*, the DNS server query tool *dig*, DNS resolve tool *host* and the depreciated but still useful DNS resolve tool *nslookup* are also part of the BIND distribution.
2. The *named.conf* configuration file. While technically this could be considered part of *named*, the *named.conf* file is the cornerstone of your DNS distribution. It defines how the daemon will respond to DNS queries from different hosts. The interfaces *named* will listen for queries about where the DNS database files are located and what DNS domains for which the server will provide responses.
3. The DNS database or "zone" files. Without getting to deep into DNS's workings, the DNS system is a global network of DNS servers that operate in an inverted tree hierarchy. At the top is what is known as root DNS servers, which act as directors to other DNS servers that are "authoritative" for their domain. That means that the servers are the master reference for domain-name-to-IP translation for their specific DNS domain. Inside of a domain, there could be more than one DNS server for redundancy and load sharing. In this configuration, one server would function as the master server for the domain and the others operate as slave servers. The slave servers talk periodically with the master server and update their DNS zone files. There are two types of zone files: forward lookup and reverse lookup. A forward and reverse lookup database file is needed for each domain for which the DNS server is authoritative.

Building BIND

The build process for BIND is quite straightforward. However, because we will be running this service on a server that straddles our secure network and our WLAN segment, we want to ensure that the server is as secure as possible. It's optimal to run *named* in a *chroot* jail, although a little more work is involved than with the standard build.

First we need to get the BIND code from isc.org:

```
root@tridant root# wget http://ftp.isc.org/isc/bind9/9.3.2/bind-9.3.2.tar.gz
--12:55:08--  http://ftp.isc.org/isc/bind9/9.3.2/bind-9.3.2.tar.gz
          => `bind-9.3.2.tar.gz'
Resolving ftp.isc.org... done.
```

DNS for a Wireless Network

Michael J. Martin

```
Connecting to ftp.isc.org[204.152.184.110]:80... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 5,302,112 [application/x-gzip]
```

```
100%[=====]  
=====>] 5,302,112    257.60K/s    ETA 00:00
```

```
12:55:29 (257.60 KB/s) - `bind-9.3.2.tar.gz' saved [5302112/5302112]
```

Then we need to extract the build files using tar:

```
root@tridant root# tar xvfz bind-9.3.2.tar.gz
```

Once we have the source downloaded and extracted into a build directory, we need to set up the chroot jail by building the jail directories and creating the named user and group.

```
root@tridant root# mkdir -m 700 /usr/local/named  
root@tridant root# mkdir -p /usr/local/named/etc  
root@tridant root# mkdir -p /usr/local/named/lib  
root@tridant root# mkdir -p /usr/local/named/dev  
root@tridant root# mkdir -p /usr/local/named/usr/sbin  
root@tridant root# mkdir -p /usr/local/named/var/named  
root@tridant root# mkdir -p /usr/local/named/var/run
```

```
root@tridant root root# adduser named -s /bin/nologin
```

With the jail built, we can move into the build directory, by running configure using the `--prefix` option so the application will install right into the jail. Then we run make and make install.

```
root@tridant root# cd bind-9.3.2  
root@tridant root# ./configure --prefix=/usr/local/named  
root@tridant root# make  
root@tridant root# make install
```

After the BIND install process is completed, we just have to finish building the cell. First, we set the proper ownership for the cell binaries and database directory:

```
root@tridant root# chown -R named /usr/local/named  
root@tridant root# chgrp -R named /usr/local/named
```

Second, we create device files for the cell:

```
root@tridant root# mknod -m 666 /usr/local/named/dev/null c 1 3
```

The final step in the cell install process is to find and replicate named binary library dependencies. Without access to the shared libraries, the named service will not run. This is because the daemon does not have system-level access to the libraries when running out of the chroot jail. To figure out the library dependencies, we need to execute the `ldd` command, which prints the shared libraries and their location.

DNS for a Wireless Network

Michael J. Martin

```
root@tridant root# ldd /usr/local/named/sbin/named
libcrypto.so.4 => /lib/libcrypto.so.4 (0xb74f2000)
libnsl.so.1 => /lib/libnsl.so.1 (0xb74dd000)
libc.so.6 => /lib/tls/libc.so.6 (0xb73a6000)
libdl.so.2 => /lib/libdl.so.2 (0xb73a3000)
libz.so.1 => /usr/lib/libz.so.1 (0xb7395000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0xb75eb000)
```

Once we have that information, we need only to recreate the lib directories structure in the jail and copy the libraries over.

```
root@tridant root# mkdir /usr/local/named/lib/tls
root@tridant root# mkdir /usr/local/named/usr/lib
root@tridant root# cp /lib/libcrypto.so.4 /usr/local/named/lib/
root@tridant root# cp /lib/libnsl.so.1 /usr/local/named/lib/
root@tridant root# cp /lib/tls/libc.so.6 /usr/local/named/lib/tls/
root@tridant root# cp /lib/libdl.so.2 /usr/local/named/lib/
root@tridant root# cp /usr/lib/libz.so.1 /usr/local/named/lib/
root@tridant root# cp /usr/lib/libz.so.1 /usr/local/named/usr/lib
root@tridant root# cp /lib/ld-linux.so.2 /usr/local/named/lib/
```

Housekeeping Tasks

Once done with the BIND install, there are a few Linux housekeeping items to take care of. Because we installed BIND outside of the standard directory trees (i.e., /usr/bin, /usr/local/bin, etc.), the manual pages are not readily accessible and the dig, host and nslookup binaries that users execute are not the new binaries, but those that came as part of the Linux install. Once the binaries have been compiled, some adjustments need to be made to the \$PATH and \$MANPATH statements. These path statements define the directories the shell searches to find the binary or manpage you have typed on the command prompt. The \$PATH statement is set in a few places; the system-wide path is set in /etc/profile. To include /usr/local/named/bin in the system-wide path, edit /etc/profile:

```
root@tridant root# vi /etc/profile
```

```
# Path manipulation
if [ `id -u` = 0 ]; then
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
    pathmunge /usr/local/named/bin <- Add this path to the end of the
list
fi

:wq
root@tridant root#
```

Alternatively, if you only want to modify your path and leave the system-wide path untouched, you can edit the .bashrc file in your home directory and add the modification path statement:

DNS for a Wireless Network

Michael J. Martin

```
root@tridant root# vi .bashrc
```

```
# .bashrc
```

```
PATH=$PATH:/usr/local/named/bin <- Add new path statement
```

```
# User specific aliases and functions
```

```
:wq
```

```
root@tridant root#
```

The \$MANPATH can also be defined in your \$HOME/.bashrc file. But the system-wide variable is set in /etc/man.config.

```
root@tridant root# vi /etc/man.config
```

```
# Every automatically generated MANPATH includes these fields
```

```
#
```

```
MANPATH /usr/share/man
```

```
MANPATH /usr/man
```

```
MANPATH /usr/local/share/man
```

```
MANPATH /usr/local/man
```

```
MANPATH /usr/X11R6/man
```

```
MANPATH /usr/local/named/man <- Add this line to the end of the list
```

```
.
```

```
.
```

```
: wq
```

```
root@tridant root#
```

Now the path statements should be all set. (You will need to log out and log back in again for the modifications to take effect.) We need to disable the old dig, host and nslookup binaries and configure the server to load named during the bootstrap process. As for the binaries, you have a few options. You can remove the package with RPM commands. This is the cleanest way, but because RPM is very interdependent, removing the package may cause other packages to break, even though you have the correct binaries installed. To uninstall the bind-utils package, do the following:

Find the package.

```
root@tridant root# rpm -qa | grep bind-*
```

```
bind-utils-9.2.2-21
```

```
ypbind-1.12-1
```

```
root@tridant root#
```

Then uninstall the package:

```
root@tridant root# rpm -e bind-utils
```

DNS for a Wireless Network

Michael J. Martin

The alternative to this is just renaming the binaries. This approach eliminates any potential complaining from RPM dependencies and allows you to use the old binaries if there is a problem. But you are left with files on your server that you will probably never use. The choice is yours.

```
root@tridant root# mv /usr/bin/host /usr/bin/host.old
root@tridant root# mv /usr/bin/dig /usr/bin/dig
root@tridant root# mv /usr/bin/nslookup /usr/bin/nslookup.old
```

Those of you who are regular readers should recall the Unix/Linux bootstrap execution scripts located in /etc/rc.d, specifically /etc/rc3.d. There are a few ways to skin the bootstrap cat. The first and most simple is to add the command to the rc.local file /etc/rc3.d/S99local or /etc/rc.local (they are the same file):

```
root@tridant root# vi /etc/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local

/usr/local/named/sbin/named -u named -c /usr/local/named/etc/named.conf
<- Add this line

:wq
root@tridant root# vi
```

This will start the service at boot, but that's about it. Now, because named is basically a database server, and database server files need to get updated from time to time, you don't want to have to reboot the server every time you make changes to the DNS zone files -- for example when you add or remove a host. If you go with the rc.local method each time you update the zone files, you will need to reload the server. This can be done using the kill command with the -HUP or hang-up option, which reloads the server files. To do this, you need to find out the process ID (PID) for named:

```
root@tridant root# ps -aux | grep named
named      9890  0.0  0.2  4472 2392 ?        S      18:46   0:00
  /usr/local/named/sbin/named -u named -c /usr/local/named/etc/named.conf
root@tridant root#
```

Then execute the kill command:

```
root@tridant root#kill -HUP 9890 <- That's the named process ID
```

The alternative to this is to build an rc loader script that you install in /etc/rc3.d. This will not only load the service during boot and shut the service down, but can also have a reload option so that each time you make modifications to the zone files you can just type:

```
root@tridant root#/etc/rc3.d/S98named reload
```

DNS for a Wireless Network

Michael J. Martin

The process will be restarted and the zone file loaded. Here is a loader script for the named implementation we have just built.

```
#!/bin/sh
# rc loader script for named

# program variables:
PID="/usr/local/named/var/run/named.pid"
BIN="/usr/local/named/sbin/named"
CONF="/usr/local/named/etc/named.conf"
PROG="$BIN -u named -c $CONF"
#

# Check that networking is up.
if [ "$NETWORKING" = "no" ]; then
    echo "no net serv"; exit 0
fi

prog=named
#
start() {
    echo "$Starting $prog "
    if [ -f $CONF ]; then
        $PROG > /dev/null 2>&1
    else
        echo "No $prog config"
        exit 1
    fi
}
#
reload() {

    echo "$Reloading $prog "
    kill -HUP `cat $PID`;exit

}
#
stop() {

    echo "$Stopping $prog "
    kill -9 `cat $PID`;exit

}
#
case "$1" in
    start)

```

DNS for a Wireless Network

Michael J. Martin

```
        start
        ;;
    reload)
        reload
        ;;
    stop)
        stop
        ;;
*)
    echo $"Usage: $0 {start|reload|stop}"
    exit 1
esac
```

Just copy that, paste it into a file called /etc/rc3.d/S98named, and you're off to the races. One last tip: If you should have any problems getting named to run and you need to see what the server is doing, the developers have provided an option to run named in the foreground and provide debugging output. To run in this mode add `-g` to the beginning of the command syntax:

```
[root@euc-ux-01 zones]# /usr/local/named/sbin/named -g -u named -c
/usr/local/named/etc/named.conf
25-Jun-2006 19:38:45.381 starting BIND 9.3.2 -g -u named -c
/usr/local/named/etc/named.conf
25-Jun-2006 19:38:45.383 loading configuration from
'/usr/local/named/etc/named.conf'
25-Jun-2006 19:38:45.387 no IPv6 interfaces found
25-Jun-2006 19:38:45.387 listening on IPv4 interface lo, 127.0.0.1#53
25-Jun-2006 19:38:45.388 listening on IPv4 interface eth0, 172.30.40.2#53
25-Jun-2006 19:38:45.389 ignoring config file logging statement due to -g option
25-Jun-2006 19:38:45.390 zone 0.0.127.in-addr.arpa/IN: loaded serial 5
25-Jun-2006 19:38:45.390 zone 75.109.156.in-addr.arpa/IN: loaded serial 5
25-Jun-2006 19:38:45.391 zone cell.outland.net/IN: loaded serial 5
25-Jun-2006 19:38:45.391 zone localhost/IN: loaded serial 5
25-Jun-2006 19:38:45.391 running
25-Jun-2006 19:38:45.392 zone 0.0.127.in-addr.arpa/IN: sending notifies (serial 5)
25-Jun-2006 19:38:45.392 zone localhost/IN: sending notifies (serial 5)
25-Jun-2006 19:38:45.392 zone 75.109.156.in-addr.arpa/IN: sending notifies (serial
5)
25-Jun-2006 19:38:45.392 zone cell.outland.net /IN: sending notifies (serial 5)
```

Now let's build the named.conf file.

The named.conf Configuration

The server we are setting up can operate either as a resolver, authoritative, or non- authoritative DNS server. A DNS resolver server, also known as a caching name server, only performs DNS lookups for external hosts. It performs host lookups or forwards DNS lookup requests to other DNS servers (which is not currently recommended by the ISC folks). An authoritative server is one that contains master DNS server records for your domain and is queried locally and externally when hostname to IP information is required for hosts that are part of the domain. The server is part of the global DNS network and is listed as part of your domain registration. An authoritative server is the definitive authority for your domain's host information. It also performs lookups for hostnames outside of the

DNS for a Wireless Network

Michael J. Martin

domain for users in the domain. A non-authoritative server performs local lookups and hosts zone information that is only locally relevant and not queried outside of the domain. A non-authoritative server is required for our WLAN environment.

The easiest way to tackle the named.conf file is to look at it in three sections:

Section 1: Server behavior and option statements

Section 2: Local host zone and root server configuration

Section 3: Local domain zone files

Section 1: Server Behavior And Option Statements

The beginning of the file starts with statements defining the directory where the binaries and supporting files are located. It includes the IP interfaces that listen for queries and the service port that binds the listener:

```
// Section One: daemon operation
options {
    directory "/usr/local/named/var/named";
    query-source port 53 ;
    listen-on { 192.168.40.2; 127.0.0.1;};
};
```

In addition to these basic definitions, it's a good idea to consider implementing access control. There are two types of access control available. The first one allows you to control who sends DNS queries to the server (and can expect a reply).

First, you need to create the user group ACLs. For an example, let's say that the WLAN supports three different user groups: Group 1 only has un-authenticated Internet http access, Group 2 has authenticated Internet http/https access, and Group 3 has unauthenticated access to Internet http/https and the private network via an authenticated VPN connection. Group classification is managed via node IP address. When you join the WLAN, you are assigned an IP address that reflects your user class. Now, to support these different groups, network infrastructure needs to exist to allow routing between these segments. However, we want the Group 1 and 2 users to use one DNS server and the Group 3 users to use another. So we create two ACLs. One is named WLAN-Private, the other WLAN-CELL. The WLAN-Private group has private Net access via VPN. The WLAN-CELL users have the http/https access.

```
acl WLAN-Private {
    172.30.40.32/28
};
```

```
acl WLAN-CELL {
    192.168.20.0/27
    192.168.30.0/24
    localhost
};
```

```
options {
Revised July 21, 2006
```

DNS for a Wireless Network

Michael J. Martin

```
blackhole { WLAN-Private; };
allow-query { WLAN-CELL; };
}
```

ACL statements are defined using a NETWORK/PREFIX statement. In addition, there are four pre-defined lists:

- Any – Matches against all IP addresses
- Localhost – Matches IP addresses used by the server
- Localnets = Matches IP addresses on any network to which the server is directly connected
- None – Matches against no IP addresses.

The second access control element is typically relevant only if your DNS server is authoritative and supports a domain that is queried by other domains.

```
Options {
    allow-transfer {
        none;
    };
}
```

This option disables the ability for an unknown user to download your DNS zone files. This may seem odd at first, since each a lookup can be performed for each host listed in the database. The difference here is that normally a user outside of the domain would only have use for one or two hosts in your domain, such as a domain's mail or http server. Allowing an unauthorized user to download your zone databases gives them a map of every host in the domain. If you have secured your environment properly, ICMP scans and the like will not reveal how many hosts are in your domain. Often when "evil doers" are staking out a target, the first thing they try is to download your DNS server's zone files. As a safety precaution, it is best to add this as part of any server configuration.

Section 2: Local Host Zone And Root Server Configuration

Now we move on to Section 2, or the "resolver" section. Every DNS server has at least one set of zone files. That set is for localhost, which is a standard set of zone files. It consists of one forward lookup database and one reverse lookup database (which we will get to in a moment) and a hints file. The hints file, properly called root.hints, is a list of all the Internet's root name servers and can be downloaded from ftp.rs.internic.net. The hints file is located in /usr/local/named/var/named. The zone database files are located in /usr/local/named/var/named/zones. The best way to get the hints file is to download it right into the proper directory using wget:

```
root@tridant root# wget ftp://ftp.rs.internic.net/domain/named.root >
/usr/local/named/var/named/root.hints
--16:22:19--  ftp://ftp.rs.internic.net/domain/named.root
           => `named.root'
Resolving ftp.rs.internic.net... done.
Connecting to ftp.rs.internic.net[198.41.0.6]:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.      ==> PWD ... done.
==> TYPE I ... done.   ==> CWD /domain ... done.
```

DNS for a Wireless Network

Michael J. Martin

```
==> PASV ... done.    ==> RETR named.root ... done.
Length: 2,517 (unauthoritative)
```

```
100%[=====
=====] 2,517
27.01K/s    ETA 00:00
```

```
16:22:21 (27.01 KB/s) - `named.root' saved [2517]
root@tridant root#
```

The named.root file is updated monthly. It's a good idea to set up a cron job to download the file every month to keep your DNS server up to date.

Depending on the DNS administrator preference, the root.hints file may also be called named.ca, named.root, or named.hints; the file names for the hints and zone files are not set in stone. They can be anything you want them to be, but the proper form to use some of the standard conventions i.e. root or hints, or in-addr (for reverse zones) and db.domain or domain.zone (for forward zone files). Here is an example of the "resolver" section:

```
\\ Root Name Servers
zone "." IN {
    type hint;
    file "root.hints";
};

\\ Localhost reverse zone
zone "0.0.127.in-addr.arpa" IN {
    type master;
    file "zones/in-addr-0.0.127";
};

\\ Localhost forward zone
zone "localhost" IN {
    type master;
    file "zones/localhost.zone";
};
```

Section 3: Local Domain Zone Files

The third section of the named.config file is the location mappings for the local zone files. If your server is simply a resolver, then you can skip on to creating the zone files section to set up the localhost databases. If you are implementing an authoritative or non-authoritative server, you will want to create entries (and database files) for the domain and sub-domains. Here is a syntax example for a local domain:

```
\\ Local Zone Files

\\ rev zone for 172.30.40.32
zone "32.40.30.172.in-addr.arpa" IN {
    type master;
```

Revised July 21, 2006

Page 10 of 14

DNS for a Wireless Network

Michael J. Martin

```
file "zones/in-addr-32.40.30.172";
allow-update { none; };
};

\\ fw zone file for wlan.outlan.net
zone "wlan.outlan.net" IN {
    type master;
    file "zones/outlan.zone";
    allow-update { none; };
};
```

Once you have created your named.conf file, all that is left to do is check it. ISC provides a named.conf syntax-checking tool called named-checkconf. The tool is located in /usr/local/named/sbin. Running it is easy; just tell the tool the location of the file you want checked. If you run the tool and get no feedback, the file is syntactically correct. If there are errors, it tells you what is wrong and where.

Creating the ZONE Files

The DNS server zone files are flat file database files comprised of different DNS resource records. There are two types of zone files: forward lookup zone files that are used for hostname to IP address translations, and reverse lookup zone files that are used for IP address to hostname translations. As you are already aware of, the most basic DNS servers needs at least two -- the Localhost.zone forward lookup zone file and the 1.0.0.127.in-addr.arpa reverse lookup zone file. The zone files are comprised of the potentially 30+ DNS resource records currently available in the IN Internet resource class. There are only two other formally assigned classes, Hesiod and Chaos, both of which are infrequently used. Of the 30+ available resource records in the IN resource class six are commonly used:

Type	IANA value	RFC
A	1 – Host address	RFC 1035
NS	2 - Name server	RFC 1035
CNAME	5 - Canonical name	RFC 1035
SOA	6 – Start of zone authority	RFC 1035
PTR	12 – Domain name pointer	RFC 1035
MX	15 – Mail exchanger	RFC 1035

Forward zone databases contain an SOA record, NS record and any number of A, CNAME and MX records. Reverse zone databases contain an SOA record, NS record and PTR records. Now that you know the makeup of the two types of zone databases, lets dive a little deeper into what the records represent, their use and format.

The A record is the most common resource record type. A resource records are used in forward lookup zone databases. It translates "human" names into IP addresses, to support forward DNS lookups. The A record is the base DNS record required for all networked computers to be supported by the DNS system. The format of the A record is fully qualified domain name, ended by a period. This is followed by the resource record class, record type, and record value. Here is an example:

```
rtr-01.outlan.net.    IN A 172.30.80.1
Revised July 21, 2006
```

DNS for a Wireless Network

Michael J. Martin

```
ns.outlan.net.    IN A 172.30.80.24
smtp.outlan.net. IN A 172.30.80.16
```

Most networks use sub-domains to translate IP subnets into a named hierarchy. Implementing this approach requires the creation of zone files (or at least a forward zone database) for each sub-domain. Here is an example A record for a sub-domain:

```
rtr-02.cell.outlan.net.net IN A 192.168.20.1
```

The NS resource record is a special record that defines the authoritative DNS server for a zone. NS resource records are used in forward and reverse lookup zone databases. These records are typically the first entries in the zone file following the SOA record. The format for the NS resource record is domain or sub-domain, record class, record type NS, and A record of the authoritative name server for the zone.

```
pre> outlan.net IN NS ns.outlan.net.
```

The CNAME resource record provides an alias function. CNAME resource records are used in forward lookup zone databases, which enables administrators to set up name pointers to physical hosts that have valid A records. For example, let's say that we want the default gateway routers on each subnet to be called "gw.sub-domain.domain.net." We would set up CNAME resource records for each sub-domain/IP subnet. The format of a CNAME resource record is the fully qualified domain name (ended by a period), followed by the record class, and record type CNAME, with the data value as the A record of the target host:

```
gw.cell.outlan.net.    IN CNAME rtr-02.cell.outlan.net
```

The SOA resource records contain the operational properties for the zone. The SOA is the first resource record of each forward and reverse zone database file. It is preceded only (optionally) with zone file time-to-live value. The SOA value is expressed in some combination of days = d#, hours = h#, minutes = m# or seconds = #. The SOA record contains the following information:

Domain Name Of Primary DNS Server For The Zone

- Email address of responsible party for the zone in the format e-mail-id.domain.com (i.e., hostmaster.outlan.net)
- Serial number for zone file. This is used by secondary DNS servers to see if the zone data has changes. The number can be any value; the secondary server will perform a zone transfer if the serial number it has is lower than the one on the master server. The most common format is to use the date day-month-year-version (i.e., 042220061)
- Refresh, how often secondary DNS servers should check to see if zone changes have been made
- Retry, how often secondary DNS servers should for changes if the refresh fails
- Expire, how long the zone information should remain valid after a refresh. The data will be deleted after this time has expired
- Minimum TTL, the time-to-live for new zone database records.

DNS for a Wireless Network

Michael J. Martin

The format of the SOA resource record starts with either the domain (i.e., outlan.net), the variable \$ORIGIN or @ (when \$ORIGIN or @ is used, the domain is taken from the named.conf zone definition). This is followed by the record class, record type SOA, and the SOA data fields. The Here is an example SOA with the zone TTL declaration:

```
$TTL    24H
;
outlan.net      IN      SOA    ns.outlan.net. hostmaster.outlan.net. (
                    5      ; Serial
                    8H     ; Refresh
                    2H     ; Retry
                    4W     ; Expire
                    1D)    ; Minimum TTL
```

The PTR resource record is the reverse of the A record. It provides the mapping of IP addresses to domain names. PTR resource records are used in reverse lookup zone databases. While the value of reverse IP address translation may seem obvious, PTR records are not required. They have become a backdoor security check for some Internet applications, but they do not always exist. This is due largely to the fact that the responsibility for PRT records fall to the owners of the IP address space, which in most cases are the ISPs providing the network service, not the owner of the domain or the domain registrar that is managing the domain. However, if you own/manage your IP address space (or maintain a private domain as we are with the WLAN solution) you are responsible for creating forward zone files containing the A records for the domain and reverse zone files containing the PTR records for the domain. The format of the PTR record is last octet of the subnet IP address or 4th.3rd.2nd.1st octet- .in-addr.arpa., record class, record type PTR, and A record for the host.

```
1      IN      PTR    rtr-01.outlan.net.
```

or

```
1.80.30.172.in-addr.arpa. IN PTR    rtr-01.outlan.net.
```

The MX resource record is a special record that defines the mail server or servers for a domain. MX resource records are used in forward lookup zone databases. The MX record maps the domain name (or sub-domain name) to a mail server or group of mail servers. When multiple servers are defined, each MX record has a preference value. The record with the lowest value is preferred. Each MX record must point to a valid A record. The format for the MX resource record is domain or sub-domain, resource record class, resource record type MX, preference, and A record of the mail server:
pre> outlan.net. IN MX 10 smtp.outlan.net. outlan.net IN MX 20 smtp.svr.outlan.net.

Here is the complete forward localhost.zone:

```
; localhost.zone
$TTL    12H
;
@      IN      SOA    ns.outlan.net. hostmaster.outlan.com. (
                    5      ; Serial
                    8H     ; Refresh
```

DNS for a Wireless Network

Michael J. Martin

```
2H      ; Retry
4W      ; Expire
1D)     ; Minimum TTL
```

```
;
                                IN      NS      ns.outlan.net.
localhost.                       IN      A       127.0.0.1
;end
```

Reverse in-addr-0.0.127 files for the named.conf configuration above look like this:

```
; in-addr-0.0.127
$TTL      12H
@          IN      SOA     ns.outlan.net. hostmaster.outlan.com. (
                                5          ; Serial
                                8H        ; Refresh
                                2H        ; Retry
                                4W        ; Expire
                                1D)      ; Minimum TTL
;
                                IN      NS      ns.mckinsey.com
127.0.0.1  IN      PTR     localhost.
;end
```

Both files utilize the @ domain declaration, so they are suitable for use as templates for creating your own forward and reverse zone files.

While this is by no means the definitive last word on implementing BIND and a DNS server, this article should have provided you with enough information to implement the needed DNS support for our WLAN environment. Next time we begin to implement the final component -- the squid proxy server -- and implement Web Proxy Auto Discovery Protocol (WPAD).