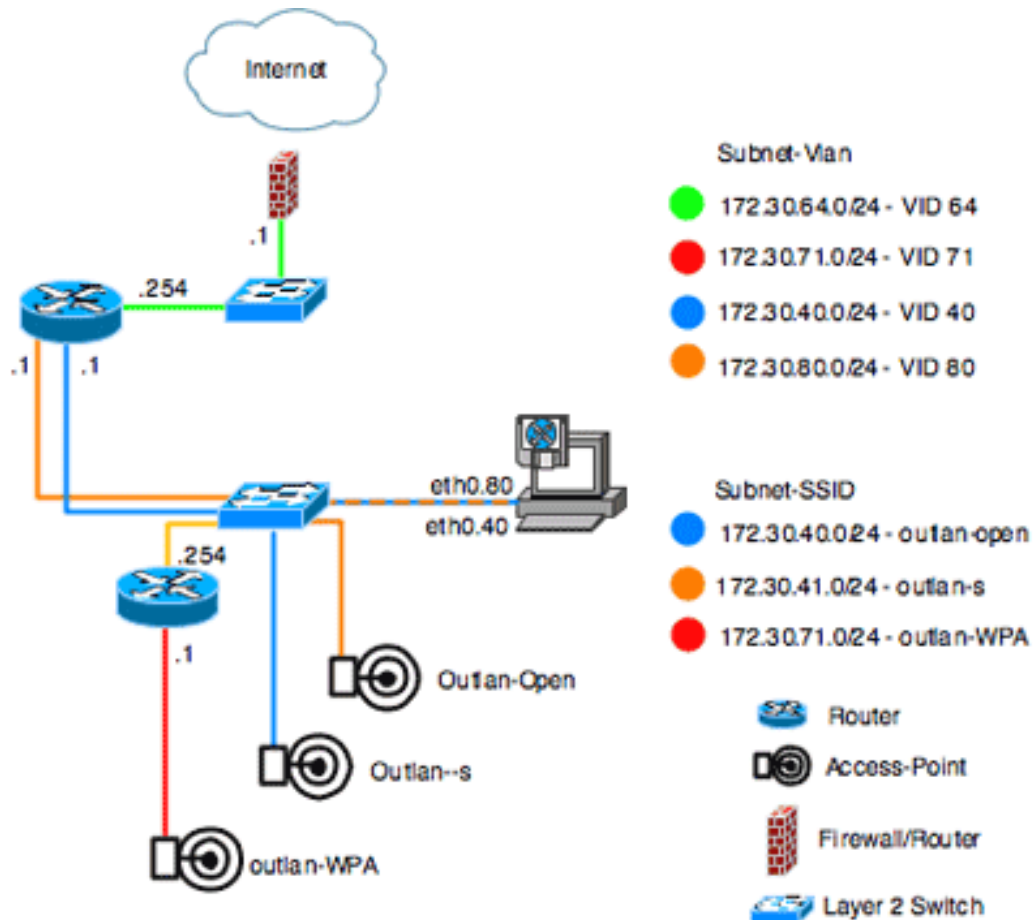


DHCP Services for WLANs

Michael J. Martin

When it comes to implementing DHCP services, we have a choice between utilizing the DHCP service provided by Cisco IOS (which I have covered implementing in previous articles) or the Internet Systems Consortium's (ISC) DHCP which can run on the Linux server. We will be using the following network diagram for our implementation discussion:



Two key features about this implementation are:

- It uses a single network interface on the Linux server, requiring the use of 802.1q
- It uses multiple gateways on WLAN/LAN segments

The first feature requires us to set up 802.1q networking on the Linux server, assuming that the server's kernel has 802.1q support built in (and it should if you have been reading this series). We must configure the server's switch port to support 802.1q trunking and the server's interface to support VLANs. The switch part is easy. First, the port's trunk encapsulation protocol needs to be set. Once that's done, the port can be set as a trunk port and the permitted VLANs can be defined:

```
outlan-sw01#config t
Enter configuration commands, one per line. End with CNTL/Z.
outlan-sw01(config)#interface FastEthernet 0/21
outlan-sw01(config-if)#switchport trunk encapsulation dot1q
outlan-sw01(config-if)#switchport mode trunk
outlan-sw01(config-if)# switchport trunk allowed vlan 40
outlan-sw01(config-if)#switchport trunk allowed vlan add 80
outlan-sw01(config-if)#^Z
```

DHCP Services for WLANs

Michael J. Martin

If you don't define the VLANs that are permitted, the switch will permit all of the VLANs configured on the switch to potentially send traffic across the port. This is not necessarily wrong, but it is better to define what traffic can traverse the port, then to leave it up to the uplink device. Here is what the final port configuration looks like:

```
outlan-sw01#sh run | begin interface FastEthernet0/21
interface FastEthernet0/21
switchport trunk encapsulation dot1q
switchport trunk allowed vlan 40,80
switchport mode trunk
!
```

Now we must configure the server. Red Hat does not by default support 802.1q. The quickest way to get VLAN interfaces up and running is to disable the bootstrap network process /etc/rc3.d/S10network by renaming the /etc/rc3.d/S10network bootstrap file to /etc/rc3.d/s10network. We can then create our own network bootstrap script, like this example:

```
#!/bin/sh
# Install this script in /etc to load VLAN interfaces as part of the
# bootstrap process using /etc/rc3.d/S99local
#
# System Variables
host=trident
domain=open.outlan.net
gateway=172.30.80.1

# App Variables
vlan=/sbin/vconfig
ifc=/sbin/ifconfig
hn=/bin/hostname
dn=/bin/domainname
rt=/sbin/route

echo "Activating eth0...."
$ifc eth0 up
$ifc eth0 mtu 1460

# Define 802.1q vlans here:
$vlan add eth0 80
$vlan add eth0 40
#
#
echo "Enabling Vlan Interfaces"
$ifc eth0.80 172.30.80.101 netmask 255.255.255.0 up
$ifc eth0.80 mtu 1460
#
$ifc eth0.40 172.30.40.6 netmask 255.255.255.0 up
$ifc eth0.80 mtu 1460
#
echo Setting hostname....
$hn $host
echo Setting Domain Name....
```

DHCP Services for WLANs

Michael J. Martin

```
$dn $domain
echo Setting Default Route....
$rt add default gw $gateway
```

The script can be loaded at the end of the bootstrap process by adding the script to the /etc/rc3.d/S99local bootstrap process file (this will also be where we start the DHCP service):

```
[root@tridant rc3.d]# vi S99local
#!/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
echo Starting Network
/etc/start-net.sh

:wq
[root@tridant rc3.d]#
```

With the server's networking out of the way, we can back to the decision about using Cisco vs. ISC DHCP. The Cisco implementation is great for most standard DHCP environments that require only the "basic" client options:

- Subnet mask (option 1)
- Router/default gateway (option 3)
- DNS server (option 6)
- Hostname (option 12)
- Domain name (option 15)
- NetBIOS name server (option 44)
- NetBIOS node type (option 46)

All of the above options are recognized by just about every DHCP client implementation available today and are generally all that is required to get a Windows, Mac OS X or Unix/Linux-based workstation on the network. That said, there are over 100 DHCP client options defined in the current IANA (Internet Assigned Numbers Authority) DHCP parameters document. The Cisco DHCP implementation supports 12 of them. So, if you are planning to support any DHCP options beyond the basics, such as Web proxy auto discovery or static route distribution, then ISC DHCP is the way to go.

Building the DHCP Server

The ISC DHCP service is freely available from the ISC Web site. (ISC is also responsible for the BIND Domain Name Service daemon.) The ISC implementation is considered by most as the "reference" DHCP distribution (as many of the developers are members of the DHCP standard working group) and is part of many core Unix/Linux distributions. The ISC DHCP service runs as a root "owned" service. This creates a potential security exploit if ever there is a vulnerability discovered with DHCP.

One way to minimize this risk is to run DHCP in a chroot "jail." This involves creating a directory tree "cell" where all of the configuration, log, and database files, along with the executables and libraries needed for the service to operate, exist apart from the rest of the system. ISC DHCP does not, however, support the chroot option as part of the native implementation.

DHCP Services for WLANs

Michael J. Martin

Luckily, programmer Ari Edelkind has written a code patch called "paranoia" that adds support for running ISC DHCP in a chroot cell. To start the build process, we first need to download the patch file using wget. This can be done from the root directory, or you can create a build directory using the following:

```
[root@tridant root]#
wgethttp://www.episec.com/people/edelkind/patches/dhcp/dhcp-
3.0+paranoia.patch
-bash: wgethttp://www.episec.com/people/edelkind/patches/dhcp/dhcp-
3.0+paranoia.patch: No such file or directory
[root@tridant root]# wget
http://www.episec.com/people/edelkind/patches/dhcp/dhcp-3.0+paranoia.patch
--08:51:27-- http://www.episec.com/people/edelkind/patches/dhcp/dhcp-
3.0+paranoia.patch
      => `dhcp-3.0+paranoia.patch'
Resolving www.episec.com... done.
Connecting to www.episec.com[69.55.237.141]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5,366 [text/plain]
```

```
100%[=====
=====>]
5,366          56.35K/s   ETA 00:00
```

```
08:51:33 (56.35 KB/s) - `dhcp-3.0+paranoia.patch' saved [5366/5366]
```

```
[root@tridant root]#
```

The current version of ISC DHCP is 3.0.3. The "paranoia" chroot patch was tested by the patch author on ISC DHCP versions up to 3.0.1.rc4. There were some security vulnerabilities discovered in ISC DHCP versions 3.0p1, 3.0.1.rc8 and 3.0.1.rc12 and 3.0.1.rc13. The patch fails on ISC DHCP versions 3.0.1.rc14 and higher. So in order to run a secure version of the ISC DHCP code and support the chroot option, we must use version 3.0.1.rc11. Just like the patch file, we can download the code using wget:

```
[root@tridant root]# wget http://ftp.isc.org/isc/dhcp/dhcp-3.0-
history/dhcp-3.0.1rc11.tar.gz
--09:36:06-- http://ftp.isc.org/isc/dhcp/dhcp-3.0-history/dhcp-
3.0.1rc11.tar.gz
      => `dhcp-3.0.1rc11.tar.gz'
Resolving ftp.isc.org... done.
Connecting to ftp.isc.org[204.152.184.110]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 853,934 [application/x-gzip]
```

```
100%[=====
=====
=====>] 853,934      267.71K/s   ETA 00:00
```

```
09:36:09 (267.71 KB/s) - `dhcp-3.0.1rc11.tar.gz' saved [853934/853934]
```

```
[root@tridant root]#
```

Uncompress and extract the tar ball:

DHCP Services for WLANs

Michael J. Martin

```
[root@tridant root]# tar xzf dhcp-3.0.1rc11.tar.gz
```

Move the patch file into the server directory of the DHCP source code directory:

```
[root@tridant root]# mv dhcp-3.0+paranoia.patch dhcp-3.0.1rc11/server
[root@tridant root]# cd dhcp-3.0.1rc11/server
```

Then apply the patch to dhcpd.c. If there are problems, the patch application will report errors. Otherwise, you should just get a simple notice and the program should exit:

```
[root@tridant root]# cd dhcp-3.0.1rc11/server
[root@tridant server]# patch dhcpd.c dhcp-3.0+paranoia.patch
patching file dhcpd.c
[root@tridant server]#
```

With the patch file successfully applied, we run the configure command from the root of the source tree directory with the patch flags:

```
[root@tridant dhcp-3.0.1rc11]# ./configure --copts "-DPARANOIA"
System Type: linux-2.2
make[1]: Entering directory `/root/dhcp-3.0.1rc11/work.linux-2.2'
Making links in common
make[2]: Entering directory `/root/dhcp-3.0.1rc11/work.linux-2.2/common'
make[2]: Leaving directory `/root/dhcp-3.0.1rc11/work.linux-2.2/common'
make[1]: Leaving directory `/root/dhcp-3.0.1rc11/work.linux-2.2'
...
[root@tridant dhcp-3.0.1rc11]#
Then build the source:
[root@tridant dhcp-3.0.1rc11]# make
...
make[2]: Leaving directory `/root/dhcp-3.0.1rc11/work.linux-2.2/dhcpctl'
make[1]: Leaving directory `/root/dhcp-3.0.1rc11/work.linux-2.2'

[root@tridant dhcp-3.0.1rc11]#
```

Once the code is built, you have two options. You can run make install, which will move the binaries, man page, and configuration files into their default install locations. Or, you can move into the work.linux-2.2 directory:

```
[root@tridant work.linux-2.2]# ls
client common dhcpctl dst Makefile minires omapip relay server
[root@tridant work.linux-2.2]#
```

You then manually install the server and man page files where you want them (for instance, into the chroot cell we are about to build). The first step is creating the user and group we want the dhcpd services to run under:

```
[root@tridant root]# /usr/sbin/groupadd dhcpd
[root@thumper root]# adduser dhcpd -s /bin/nologin
```

Then build the directories:

```
[root@thumper root]# mkdir -m 700 /usr/local/dhcpd
```

DHCP Services for WLANs

Michael J. Martin

```
[root@thumper root]# mkdir -p /usr/local/dhcpd/etc
[root@thumper root]# mkdir -p /usr/local/dhcpd/lib
[root@thumper root]# mkdir -p /usr/local/dhcpd/dev
[root@thumper root]# mkdir -p /usr/local/dhcpd/sbin
[root@thumper root]# mkdir -p /usr/local/dhcpd/var
[root@thumper root]# mkdir -p /usr/local/dhcpd/var/state
[root@thumper root]# mkdir -p /usr/local/dhcpd/var/state/dhcp
[root@thumper root]# mkdir -p /usr/local/dhcpd/var/run
```

Now we need to create some special files and copy the libraries the daemon needs in order to run. Let's start with the device files:

```
[root@thumper root]# mknod -m 666 /usr/local/dhcpd/dev/null c 1 3
[root@thumper root]# mknod -m 666 /usr/local/dhcpd/dev/random c 1 3
```

Now we copy the /etc/localtime file to /usr/local/dhcpd/etc so the syslog messages have the correct time stamp information.

```
[root@thumper root]# cp /etc/localtime /usr/local/dhcpd/etc/localtime
```

Once the device files are complete, we need to figure out what libraries we need and copy them into the chroot directory tree:

```
[root@tridant root]# cd /dhcp-3.0.1rc11/work.linux-2.2/server
[root@tridant server]# ldd dhcpd
      libc.so.6 => /lib/tls/libc.so.6 (0x4001e000)
      /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
[root@tridant server]# cp /lib/ld-linux.so.2 /usr/local/dhcpd/lib/
[root@tridant server]# mkdir /usr/local/dhcpd/lib/tls/
[root@tridant server]# cp /lib/tls/libc.so.6 /usr/local/dhcpd/lib/tls/
[root@tridant server]# ln -s /usr/local/dhcpd/lib/tls/libc.so.6
/usr/local/dhcpd/lib/libc.so.6
```

Now, let's install the daemon and create the dhcpd.leases file:

```
[root@tridant server]# cp dhcpd /usr/local/dhcpd/sbin/dhcp-3.0.1rc11
[root@tridant server]# /usr/local/dhcpd/var/state/dhcp/dhcpd.leases
```

Now that all of the files are in place, we need to set the user and group the permissions correctly:

```
[root@tridant server]# chgrp -R dhcpd /usr/local/dhcpd/
[root@tridant server]# chown -R dhcpd /usr/local/dhcpd/
```

With the daemon installed and the chroot jail built, all that's left to do is get the service started and set to run when the system boots. We also must set up our dhcp scopes in the dhcpd.conf file. But those are projects for next time, so stay tuned.