**Why bother with tftp?**

Many network devices such as Cisco routers and switches use tftp in order to download their IOS config updates. tftp can also be used for network based installs or for booting up diskless systems. Knowing how to setup a tftp server comes in quite handy when circumstances like these arise.

**Getting started . . .**

**The Network Topology**

Let’s say we’re dealing with a private network– 192.168.100.0/24. We’ll designate our tftp server and tftp test client as 192.168.100.5 and 192.168.100.105 respectively. You will need superuser privileges on both your server and client in order to successfully perform all of these commands.

**Get the Necessary Packages**

Log on to 192.168.100.5 and download the necessary programs; make sure they survive reboots:

# yum install tftp-server xinetd

You have to reinstall xinetd (even if you already have xinetd)

# chkconfig tftp on

# chkconfig xinetd on

**Why Xinetd?**

**Use command systemctl instead of service if you are using systemd. Actually if you use service command, service command will call systemctl command for you. So you don’t need to use systemctl command directly**

Why the need for xinetd? The xinetd daemon is a “super-daemon” or “super service” that listens for connection requests on behalf of other daemons and services. You can launch xinetd once and have it wake up other intended services as needed. Xinetd brings efficiency and security by being a single service that runs as needed– as opposed to having multiple, dormant services running needlessly in the background for most of their run time.

In this post, xinetd will be used to listen for any tftp requests that come in from clients. When the server receives a request, xinetd will launch tftp with the necessary options so that files can be downloaded by the tftp client as requested. Yes . . . you can run tftpd without xinetd; however, the xinetd config file keeps all the important tftpd options nice and tidy for you as we can see here:

After installing tftp-server, file tftp is automatically created in /etc/xinetd.d

**# cat /etc/xinetd.d/tftp**

...

service tftp

{ disable = yes

socket\_type = dgram

protocol = udp

wait = yes

user = root

server = /usr/sbin/in.tftpd

server\_args = -s /var/lib/tftpboot

per\_source = 11

cps = 100 2

flags = IPv4

}

That’s the default xinetd config file for the tftpd service. The default settings work just fine and the config file already comes configured once you download xinetd and tftp-server via yum.

If you still insist on running tftpd without xinetd, consult the man pages for in.tftpd. You’ll need to know how to use the correct options in order to get the tftpd server listening in stand-alone mode (that is to say, listening without xinetd).

**System Prep**

**Turn up the necessary conntrack module in iptables:**

If you’re running iptables on your system, you will need to enable the connection track module for iptables. This is not an optional step; Iptables will not allow tftpd to work properly without turning up the connection track module.

Why?

Initial tftp connections occur over port 69 by default; however, port 69 is not where the file transfer actually happens. The file transfer actually completes across the higher numbered, unprivileged ports. Unprivileged port ranges include port 1024 and above– making port 69 a privileged port. Privileged ports are designated for service transactions which require the permission levels of a privileged user account. Once the initial tftp traffic is established, that traffic migrates over to the unprivileged port ranges. The default iptables rules will lose track of your tftp session during this port migration and actually start blocking your intended file transfer as a result. By enabling the connection track module, iptables will temporarily and dynamically open up the necessary unprivileged port ranges until your transaction is complete.

I suspect many of the headaches generated by first-time tftp server set-ups are due to overlooking the implications of the connection track module.

Now that we see how important connection track module is, let’s set it up; We must edit the iptables configuration file as follows:

# vi /etc/sysconfig/iptables-config

Change the line:

IPTABLES\_MODULES=""

To:

IPTABLES\_MODULES="ip\_conntrack\_tftp"

If you happen to have a module already configured here, just append a space along with your second module within the quotes like so:

IPTABLES\_MODULES="ip\_conntrack\_ftp ip\_conntrack\_tftp"

Save your changes.

**Open the tftp port, or this will be a really short trip . . .**

Remember, port 69 still needs to be allowed on our server so that clients can connect. We’ll open up port 69 to our whole private network:

# iptables -I INPUT -p udp --dport 69 -s 192.168.100.0/24 -j ACCEPT

# service iptables save

# service iptables restart

If you only want to open up port 69 to your specific client, then substitute the CIDR notation with the specific IP address of your tftp client– which in this post, we have designated as 192.168.100.105.

That restart command of iptables above is to force the firewall to re-read configuration file and recognize the conntrack module you just activated.

Make sure port 69 UDP is also open on your client device; earlier, we designated our client as 192.168.100.105. Also, ensure you have a tftp client installed on 192.168.100.105:

# yum install tftp

Running the above command on your client device should do the trick– assuming your tftp client is also a properly registered CentOS/RHEL OS.

On our *client* at 192.168.100.**105**, ensure UDP traffic is allowed from the server:

# iptables -I INPUT -p udp -s 192.168.100.5 -j ACCEPT

**Avoid the common “gotchas” people experience with tftp**

Resist the temptation of changing the default settings before your first test. The defaults really do work in CentOS/RHEL 6. The problem usually has to do with SELinux or settings in iptables. Making arbitrary changes to /etc/xinetd.d/tftp, changing permissions on /var/lib/tftpboot, or changing file ownership of your download files only makes things worse; Keep everything default.

Again, make sure firewall settings allow for tftp for both the server and your client.

**Service Sanity Check**

The tftp daemon listens via xinetd. So, start xinetd up rather than in.tftpd:

# service xinetd start

Verify tftp is listening with netstat:

**# netstat -tulp |grep tftp**

udp 0 0 \*:tftp \*:\*

1142/xinetd

If your output is blank, then tftp isn’t listening. Make sure xinetd is running; if you decided to run tftpd without xinetd, make sure you’re using all the proper options to launch the tftp server into daemon mode (aka stand-alone mode, running apart from xinetd). Consult the man pages on in.tftpd for those details.

Another way to verify that tftpd is listening is with the lsof command:

**# lsof -i:69**

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

xinetd 1142 root 5u IPv4 65865 0t0 UDP \*:tftp

The output will be undeniable; whether you chose to use xinetd or not– your service isn’t running if the output is blank.

Assuming your service is running, let’s create a test file on the server. The file needs to be in /var/lib/tftpboot:

# vi /var/lib/tftpboot/test.txt

Add some text and save the file.

Now log on to your *test client* at 192.168.100.**105** and try to download the test file with a tftp client:

# tftp 192.168.100.5

tftp> get test.txt

tftp> quit

# cat test.txt

The contents of your test file should output to your screen. That means everything worked.

If the file contents was blank and you only ended up with an empty file– it’s time to troubleshoot.

**Troubleshooting Tips:**

**Time to raise the tftp debugging levels:**

# vi /etc/xinetd.d/tftp

Change:

server\_args = -p -s /var/lib/tftpboot

To read:

server\_args = -p -svv /var/lib/tftpboot

Here, we’re adding in the verbose flag so that more detailed output will be sent to /var/log/messages. Xinetd uses the server-args directive to pass options to the tftp service as it opens new connections. While error messages from tftpd can be a little bit cryptic, your favorite search engine should help you figure out what most of these errors really mean. The problem will normally point you back to poorly configured firewall rules at either the server or the client, SELinux issues on the server, or firewall rules imposed on your private network by a router or switch.

Now that you’ve set the verbosity levels for debugging, reload xinetd:

# service xinetd restart

Now, you’ll get some output in /var/log/messages about why tftp might not be working as you wished.

Also, use tcpdump to help you see what’s happening. See if the client and server are actually talking or not.

**SELinux?**

Lastly– but importantly– is SELinux enabled on your server? Instead of totally disabling it, set SELinux into permissive mode first:

# setenforce 0

Permissive mode allows SELinux to continue to log and alert without enforcing the security policies and getting in the way of your services. This way, you understand why SELinux is blocking your activities. Having that knowledge will help you run your services smoothly while keeping SELinux enabled.

Now . . . try your file transfer again. If it worked this time, then SELinux was blocking your transaction. Otherwise, your obstacle is elsewhere in this particular case.

**SELinux File Context and Restorecon**

Even with SELinux disabled, you should still determine that your files have the correct file context. Make a quick check for the SELinux file context of your files in /var/lib/tftpboot.

# ls -lZ /var/lib/tftpboot/

Examine the output. The Z flag in the ls command outputs the SELinux file context of any listed file or directory. Any file missing the tftp file security context will be barred by SELinux when download attempts are made. So in the output, make sure each file has: tftpdir\_rw\_t:s0 in the output; or else, SELinux will block access to that file.

Of course, we’ll need to fix that . . .

# restorecon -R /var/lib/tftpboot

To understand the point behind this command, you have to realize that SELinux has a predetermined file context that it looks for based on the parent directory. Running the restorecon command will cause all the files in the tftpboot directory to inherit the proper file context for tftp files. This problem with incorrect file context commonly happens when new files are moved into the tftpboot directory for download. If you move a file from– say– your home area to tftpboot— expect that file to be missing the correct file context. Running the above restorecon command will easily fix this issue.

Now, put SELinux back into Enforcing mode and watch tftp work:

# setenforce 1

Now, try your download attempt again.

Oh? You don’t have restorecon on your server? Download it by running:

# yum install policycoreutils

Restorecon is part of a suite of SELinux troubleshooting tools which are bundled together in the policycoreutils package. You’ll download a lot more than just the restorecon program. However, if you’re serious keeping SELinux enabled, you’ll need certainly need the policycoreutils package.

**Recap**

The tftp protocol is a handy way to perform network installs, update images on Cisco devices, or boot diskless workstations. The tftp server is easy to set up if you understand how to avoid certain pitfalls such as poor firewall configurations, omitting the connect track module, keeping default settings, and taking advantage of xinetd. Don’t forget about SELinux and also remember that your client firewall also needs to allow tftp traffic, too. Remember these points and setting up tftp should be a snap.