Server System Management - Linux

Lab 09: Network Boot

INHOUD

[1 Lab 09: Network boot 3](#_Toc156995545)

[1.1 Introduction 3](#_Toc156995546)

[1.2 Step 0 : Preparing the playground. 3](#_Toc156995547)

[1.3 Step 1 : Save us, DNSMASQ ! 3](#_Toc156995548)

[1.4 Now that we’ve found our netboot…. 4](#_Toc156995549)

[1.5 Start up the Network Installer, Jim ! 4](#_Toc156995550)

[1.6 I want a true diskless workstation 4](#_Toc156995551)

[1.7 Setting up your root filesystem 5](#_Toc156995552)

[1.8 Booting without NFS root filesystem 7](#_Toc156995553)

# 

# Lab 09: Network boot

## Introduction

When a server/pc boots, it locates a boot-loader or a bootable kernel on its hard disk, or on another medium such as a floppy disk, a CD-ROM, or a USB device. Most network cards, however, also have to capability of being used as a bootable medium : they contain a (ROM) chip that contains instructions to boot from a network server.

Using such a network card it is possible to boot a pc by loading boot code from a network server. This can be used to setup diskless clients (terminals, thin clients, ... ), to bootstrap terminals, or to initiate an operating system setup (by loading a remote installer instead of an operating system).

In this lab, we'll explore the basic concepts of network booting, network installation, and diskless clients. You can build on this to create your own implementation and host all your operating systems installation media and/or actual runtime environments for ANY Environment and Hardware (Linux, Windows, PC, MAC, even Raspberry Pi) on 1 central server

## Step 0 : Preparing the playground.

**(I’ve noticed that you should set all network interfaces we’ll be using on a separate vm subnet (the ens36 dnsmasq listens on and the one nic on your pixie server). Then you can keep the vmware dhcp server running. But if it interferes, you’ll have to turn the vmware dhcp off. If you DO have to turn it off, you’ll be without a VMWARE DHCP Server for a large part of our lab, so set the lease of the vmware dhcp server to 24 hours or so. Or work on the console.)**

We will need to tweak our Prime server so that it provides a custom dhcp configuration, serves a bootable kernel over tftp, and, especially for diskless clients, provides a network file system or a ramdisk with a root filesystem to the clients.

For a client, you’ll need another VM capable of PXE network boot.

1. For that, create a new vm in vmware with the smallest possible hdd (0,001GB) with a single Network Card. Put the Network Card on the same isolated vmware subnet as your prime’s ens37. Call the vm **familyname-firstname-pixie.** Do NOT install an OS (select you’ll install it later. But that’s a lie, you never will 😊 )

A screen shot of a computer

Description automatically generated

## Step 1 : Save us, DNSMASQ !

1. Look back at your dnsmasq.conf. In addition to the usual dhcp configuration (address, netmask, ...), we will need to add some of the following options :

path to file,server name,next server (tftp server address)

1. the 2 server parameters are not necessary if the boot server is the same machine as the dhcp server (as It is in our setup); they refer to the tftp server where the client will get its bootloader or operating system from. The path to the file corresponds to the tftp location of the boot loader - we'll come to that in the next steps.
2. **Edit the dnsmasq conf so that the tftp functionality is enabled and hosts up the bootloader at/srv/tftp/pxelinux.0** (yes, dnsmasq doesn’t only DNS & DHCP, it also offers a t(rivial)ftp server. tftp (trivial file transfer protocol) is a trivial file transfer protocol (duh). It's used for straightforward file transfers without any bells or whistles. DNSMASQ is so awesome.)

tftp-root=/srv/tftp

A black background with blue letters

Description automatically generated

enable-tftp

A black screen with blue letters

Description automatically generated

1. **QUIZ** : with which option in dmsmasq.conf did you specify the NAME of the pxelinux.0 boot loader ?

dhcp-boot

A screen shot of a computer

Description automatically generated

1. By default, tftp doesn't do any user authentication. Let’s make it a little more secure by enabling the tftp-secure option. By doing that,you’ll need to make sure the files you want tftpd to serve are owned by a specific user. **QUIZ : which user ?**

A screen shot of a black background

Description automatically generated

dnsmasq

1. Create the directory /srv/tftp where we will be serving files from.

sudo mkdir /srv/tftp

1. Change the owner of /srv/tftp to the special user we mentioned just before. *You’ll have to do the latter for EVERY FILE you’ll put in that folder in the remaining exercise !!*

sudo chown dnsmasq:users /srv/tftp

1. Restart dnsmasq (it should give no errors)

sudo systemctl restart dnsmasq.service

1. Now let’s see what we have so far. Turn on your **familyname-firstname-pixie** (called pixie from now on) (upload **SCREENSHOT of your identifiable vm screen**)

A screenshot of a computer

Description automatically generated

1. LOOK at what you’re seeing. Pixie has no HDD, so it tries to network boot. You'll see it gets an IP address from your prime in the 10.10.\* range, but then it fails with "tftp timeout" because we've instructed it to download a file from tftp, but we haven't set that up yet. It does show that dhcp is working and the boot options are passed to the client. We’re already this far.

From the very beginning the setting was awfully explained. I tried to create a new DHCP NAT service (i.e. separate one) in VMWare, but then all I had to do was just disable DHCP and then reload and see until it works. So disabling the default DHCP was the way.

## Now that we’ve found our netboot….

1. Now, we’ll need something to actually boot. To keep things simple, we take the Debian netinstaller. This is a set of installer files meant for network installs. We'll simply put these in /srv/tftp where the tftp daemon will serve them to the client when it tries to boot from the network
2. You can find these files at http://ftp.nl.debian.org/debian/dists/bookworm/main/installer-amd64/current/images/netboot/. On your Prime, cd to /srv/tftp Now download the download link in your vm with wget. and then unpack the netboot.tar.gz file at /srv/tftp

cd /srv/tftp

wget <http://ftp.nl.debian.org/debian/dists/bookworm/main/installer-amd64/current/images/netboot/netboot.tar.gz>

tar -xvf netboot.tar.gz

1. **QUIZ : while in /srv/tftp/ and using ONLY the tar command, untar AND unzip netboot.tar.gz. do it verbosely. Which statement did you use ?**

sudo tar -xzvf netboot.tar.gz

not to forget that I set the owner for the files to the dnsmasq:users for tftp and the files inside too.

1. Use ls to see what you’ve extracted. Note the pxelinux.0 file. This is a boot loader. The file pxelinux.cfg/default contains boot parameters for the boot loader. It will inform the boot-loader about what kernel to load, with which parameters, and where to find it. pxelinux is a part of the syslinux framework, a universal linux boot loader.
2. While your're at it, make sure that these files are world-readable (r’s for everyone, should be so already normally) (and remember something else they should be ownershipwise…). Restart dnsmasq afterwards

not to forget that I set the owner for the files to the dnsmasq:users for tftp and the files inside too.

sudo systemctl restart dnsmasq.service

## Start up the Network Installer, Jim !

1. Reboot your pixie machine. It’ll find something now ! It’ll try and start the Debian installation from the tftp location. Decline it’s kind offer to install Debian, but appreciate that’s you’ve already managed to potentially install an OS on machine with no HDD, USB or DVD attached.
2. So don’t install, just power down the pixie machine again, but first UPLOAD A SCREENSHOT of the installation startup screen, showing your pixie machine identifiable in vmware (**SCREENSHOT**)

A screenshot of a computer

Description automatically generated

I had to redownload all the files, change the ownership to simply dnsmasq, so that the tftp and the files in it has ownership set to dnsmasq and the group to root. also the files have 755 permissions.

A screenshot of a computer program

Description automatically generated

fun fact – after you reboot, you need to systemctl restart dnsmasq otherwise it does not give the file.

## I want a true diskless workstation

1. We’ve seen how we can set up a network boot server consisting of a dhcp server and a tftp server. We've set this up to serve a syslinux boot loader to a client, so it can boot from a network server. The example we used booted the Debian installer.
2. It's also possible to boot an entire operating system this way, not just an installer. This allows you to run a diskless client that you may want to use as a thin client for server-based computing. An other application might be that you just boot the operating system for the server (for centralized OS management), but run applications locally, or use a local filesystem for specific user files.
3. What do we need to do this ?

a network boot server with dhcp and tftp

a pxe boot loader configured to boot a (linux) kernel

a suitable kernel and initial (RAM) filesystem to support it

a root filesystem to be used after the kernel has booted

1. **QUIZ** : which of these do we HAVE already available ?

a network boot server with dhcp and tftp

a pxe boot loader configured to boot a (linux) kernel

a suitable kernel and initial (RAM) filesystem to support it

## Setting up your root filesystem

1. The root filesystem is the filesystem the client will use after the operating system has booted. This could be any filesystem you can mount after the kernel has booted, but for diskless workstation, there are roughly 2 ways to accomplish this.
   1. The most common approach is to provide a filesystem via NFS (Network File System). This means you have to set up an nfs server that serves a complete filesystem for the clients.
   2. A completely different, and far less common approach is to let the diskless workstations use a RAM drive. When a linux kernel boots, it initially uses a RAM drive with a minimal file system. This filesystem is extracted from an archive, the 'initrd' (initial ram drive). The initrd contains the files the operating system needs to boot, and is abandoned when the root filesystem is mounted (remember OS Concepts ?).Now, it is possible to modify the initrd or create a custom archive, and let the kernel use that as root file system. In doing so, the entire root file system is served to the clients as an archive file (via tftp), and there is no need to set up nfs. The size of the initrd, however, is limited, so this will only work for minimal client systems, e.g. a client that only needs to start terminal sessions on the server.But it is very neat to pull off.
2. We'll look into the nfs approach first, as this is more common and universal.
3. Create, on Prime, a directory that will contain the client root filesystem : /srv/clientrootfs. Then, set up nfs on the server so that it exports this directory ONLY to the clients in the network range 10.10.XX.1 to 10.10.XX.254 (check the nfs lab how to do this). No need to do any client configuration here. Specify that this directory will be exported 'no\_root\_squash' on the ip range of your prime’s dhcp server. The kernel will need root access to these files. Include 'no\_subtree\_check' to increases boot speed because the wait for nfs is shorter. And the filesystem needs to be read/write, because you’ll want users on your pixiemachine to actually WORK on it.

**QUIZ : what line did you add in your /etc/exports config file ?**

mkdir -p /srv/clientrootfs

/srv/clientrootfs 10.10.206.1/24(rw,no\_root\_squash,no\_subtree\_check)

1. Activate the export.

sudo exportfs -ra

sudo exportfs -v

A screenshot of a computer program

Description automatically generated

1. Now we will make sure '/srv/clientrootfs' on the server will become '/' to the pixie client.
2. The problem is how to get a working operating system inside /srv/clientrootfs. There are a number of ways to do this : You can copy or rsync an existing filesystem into /srv/clientrootfs (*this is interesting… This means you could relatively easily migrate any existing local storage-based client setup to a netbooted diskless setup). Y*ou can use debootstrap to create a (minimal) Debian system inside /srv/clientrootfs. You can extract a live CD to /srv/clientrootfs.
3. In our lab, we'll use **debootstrap**. First, install debootstrap with apt install debootstrap. Read up on its man page. Debootstrap basically allows you to create a bare-bones Linux distro. For our lab, run debootstrap with parameters to indicate into which directory the system will be installed (/srv/clientrootfs), and the mirror server that provides the required packages (just get a **stable** version from the http://deb.debian.org/debian/ repository)
4. **QUIZ : which debootstrap statement did you use ?**

sudo debootstrap stable /srv/clientrootfs http://deb.debian.org/debian/

1. So '/srv/clientrootfs' on the server will become '/' to the clients as we said before. In order to modify this ‘/’’s behavior, we use the 'chroot' command on Prime to descend into /srv/clientrootfs and work in it as if it were /. This allows you to run standard tools we’ve learned before to modify what will become the root filesystem to the pixie client.
2. Explore the man command for chroot and then chroot to /srv/clientrootfs. This will actually make your Prime vm behave ‘kinda’ as if you’re running it as if /srv/clientrootfs is the / location.

sudo chroot /srv/clientrootfs

1. One of the things you need to add is a kernel. You could add any additional software, edit a configuration file such as /etc/apt/sources.list, create extra user accounts .... :
2. For now, lets do just this :
   1. Install kernel (linux image) using apt. (You may have to use an apt-helper package to find the correct (latest) image file for the amd64 architecture)

apt-get update

apt-get install linux-image-amd64

* 1. Make sure the pixie machine gets the hostname familyname-firstname (adapt 2 files)

nano /etc/hostname

nano /etc/hosts

* 1. Create a user pixie-yourfirstname (make sure you choose a password that’s identical in azerty/querty!)

adduser pixie-serafim

1. exit chroot when you’re done.

exit

1. installing linux-image has installed a kernel and created an initial ramdrive image in /srv/clientrootfs/boot/. These need to be copied to /srv/tftp (rename them there to vmlinuz and initrd.img) so that the tftp server can serve them to the diskless pixie machine when it tries to boot.

sudo cp /srv/clientrootfs/boot/vmlinuz-5.10.0-0.bpo.3-amd64 /srv/tftp/vmlinuz

sudo cp /srv/clientrootfs/boot/initrd.img-5.10.0-0.bpo.3-amd64 /srv/tftp/initrd.img

don’t forget chown to dnsmasq

and chmod 755

1. Next, the pxelinux.0 bootloader needs to be set up to boot this kernel. We'll reuse the pxe bootloader we copied out of the debian netinstall CD, but replace it's configuration file.
2. Remove obsolete links to pxelinux.0 and pxelinux.cfg

sudo rm -f /srv/tftp/pxelinux.0

sudo rm -rf /srv/tftp/pxelinux.cfg

1. copy debian-installer's pxelinux.0 to /srv/tftp

sudo cp /srv/tftp/debian-installer/amd64/pxelinux.0 /srv/tftp/pxelinux.0

1. Make sure that these files (at least) now exist in your /srv/tftp directory and that they are all owned by dnsmasq !

initrd.img ldlinux.c32 (this is actually a link, but should work ok) pxelinux.0 pxelinux.cfg (recreate this dir) version.info vmlinuz

1. The pxelinux.cfg/default file tells pxelinux.0 which kernel to boot, and what parameters to pass to the kernel. Create the file so it looks like this :

*DEFAULT netboot*

*SAY Netbooting my familyname-firstname-Pixie*

*TIMEOUT 0*

*PROMPT 0*

*LABEL netboot*

*KERNEL vmlinuz*

*APPEND initrd=initrd.img ip=dhcp –*

*A computer screen shot of a computer

Description automatically generated*

IMPORTANT NOT TO FORGET the chown dnsmasq and chmod 755!!!

1. Power on the diskless pixie vm and see if it boots. If all is well, you'll see the kernel boot, but it will hang or fail as it can't mount a root filesystem (you may be dropped to a busybox emergency shell). That's because we haven't told it where to find a root filesystem. Adapt pxelinux.cfg/default as follows to fix this.

*DEFAULT netboot*

*SAY Netbooting my familyname-firstname-Pixie*

*TIMEOUT 0*

*PROMPT 0*

*LABEL netboot*

*KERNEL vmlinuz*

*APPEND initrd=initrd.img ip=dhcp root=/dev/nfs nfsroot=yourprime’s10.10.XX.YY\_address:/srv/clientrootfs rw –*

*A computer screen shot of a computer

Description automatically generated*

1. Now, reboot the diskless client. You'l see it boot, setup its network interface, and load the nfs filesystem. Now login with the pixie-user account you created earlier. (hope your password is working. If not, you can reset it using chroot on Prime’s /srv/clientrootfs and using the passwd command there) **Congratulations** ! You’ve just launched your diskless pixie. Now upload a screenshot of your vm machine, logged in with user pixie-user executing the df -h command (screenshot)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Now create a file testfile in your pixie-user’s home directory (while logged on as pixie-user on your pixie-machine) with nano and enter some text there. Save. Now think a moment… **QUIZ : How could a snooping root on Prime see what you’ve just entered there ?**

cat /srv/clientrootfs/home/pixie-serafim/testfile

because we have a nfs share.

**Note :**

We've established that we can create a diskless workstation that boots of a network server and uses a network filesystem exported by an nfs server. Note that the source of that client system can be modified by chrooting into the exported directory on the server.

The way we've set things up here, any pixie-like client will get to use the same filesystem. You may want to create additional/special nfs exports for /home or other directories. As Linux uses a unified directory tree where you can mount several filesystems (remember OS Concepts ?), it is equally possible to mount other filesystems as well. For instance, you may have a separate nfs server that exports user home directories that you'll mount to /home, or mount a /local/usr/bin from the workstation's hard disk to provide local programs in combination with the operating system provided by the server. Other interesting directories to mount separately (nfs or local) are /usr or /var.. You can easily modify[/srv/clientrootfs]/etc/fstab to accomplish this.

Because all pixie-like clients share the same root filesystem, they'll have the exact same configuration. The downside is that things you want to be unique, such as host names, aren't anymore. You'll want to work around this, eg by creating separate filesystems for every client, or by using dnsmasq extensively (things such as reservations based on mac address, assigning hostnames by dhcp or from a combination of /etc/ethers and DNS records), you possibly need separate kernels for different hardware, ....),

If not all of the diskless clients have the same hardware, you may have to provide separate kernels and initrd images, and separate configuration files for software that is hardware-specific (sound, video, ...), and configure dnsmasq and pxelinux accordingly. pxe can also be set up to find a configuration file by client ip address, MAC address, or be configured to present a boot menu to let the user chose a boot configuration.... We’ll play around with this when we try :

## Booting without NFS root filesystem

1. When the kernel boots, it initially uses a ramdrive as its filesystem (remember OS Concepts ?). This filesystem is later replaced by a more permanent filesystem. In the previous configuration, we used nfs to provide that “final” root filesystem to diskless clients. But it is also possible to keep using the initial ramdrive.

The difficulty in this approach lies in the fact that you will need to create a custom initrd that can be used as initial filesystem as well as root filesystem, with at least the minimal functionality to provide a usable system to the user.

Another problem is that the initrd needs to be reasonably small. Too large a file and the tftp server will refuse to serve it.

1. To get a feel for diskless clients without nfs root filesystems, we will have a look at Damn Small Linux (dsl).
2. Let’s modify our pxe and tftp configuration so that the client will boot a dsl desktop from RAM. Start by obtaining a DSL iso and put it in root user’s home folder /root

wget http://distro.ibiblio.org/damnsmall/current/dsl-4.4.10-initrd.iso

1. Now did you know you can mount an iso file directly as if it is a cdrom under linux ? Mount the iso file as a loop device of type iso9660 under /mnt
2. **QUIZ what mount command did you use ?**

sudo mount -o loop -t iso9660 /root/dsl-4.4.10-initrd.iso /mnt

1. Now copy /mnt/boot/isolinux/minirt24.gz and /mnt/boot/isolinux/linux24 to /srv/tftp (remember the ownership “thing”!)

sudo cp /mnt/boot/isolinux/minirt24.gz /srv/tftp/initrd.img

sudo cp /mnt/boot/isolinux/linux24 /srv/tftp/vmlinuz

Now let’s make sure our Pixie Client boots this way. You COULD set up a menu structure in pxeboot, but that’s pretty advanced and requires user interaction. Let’s instead set up our PXE Server to recognize our Pixie’s MAC Address. How do we do that ?

1. Copy pxeboot.cfg/default to pxeboot.cfg/01-yo-ur-ma-ca-ad-ress (your pixie client’s mac address, but with hyphens instead of “:” between the hex’s and preceded by 01- (ARP for ethernet) and preserve the ownership while copying.

A computer screen with white text

Description automatically generated

1. Now edit that macaddress-like file in /pxelinux.cfg to look like this

DEFAULT dsl

*SAY Netbooting my familyname-firstname-Pixie via MAC-address recognition*

TIMEOUT 0

PROMPT 0

LABEL dsl

KERNEL linux24

APPEND initrd=minirt24.gz ip=dhcp ramdisk\_size=100000 init=/etc/init

BOOT\_IMAGE=knoppix --

1. Now (re)Boot your pixie vm. See what happens ? The pxe server recognizes your Mac address and uses the appropriate configuration instead of the default one.

NOTHING JUST HAPPENS IT IS IMPPOSSIBLE TO LOAD IT

I have spent 2 hours trying to first download, then copy the files into the /srv/tftp. Then it actually was still loading default configuration (and that like 1 out of 10 times). After that, I had to countlessly restart dnsmasq while the pixie was booting so that it would actually get the tftp server and then get ONLY the default configuration. I gave up and decided to get the default configuration to do the other configuration work because I just cant, it just does not want to work even barely.

1. The trick is that there is a (small) knoppix Live-CD image in /dev/cdrom of the intitrd filesystem, which is run by the pixie client as if it was a local CD.
2. Creating this kind of setup will result in a fully functional fairy-diskless client, that will keep working, even if you yank the network cable out now because everything is in Pixie’s ram.
3. If you want interaction, you’ll need an actual usb-connected mouse or select a ps/2 style mouse (that’s emulated pretty well on vm) 😊
4. Upload a screenshot of a game running under knoppix in your identifiable vmware pixie VM (**SCREENSHOT**)

A screenshot of a computer

Description automatically generated