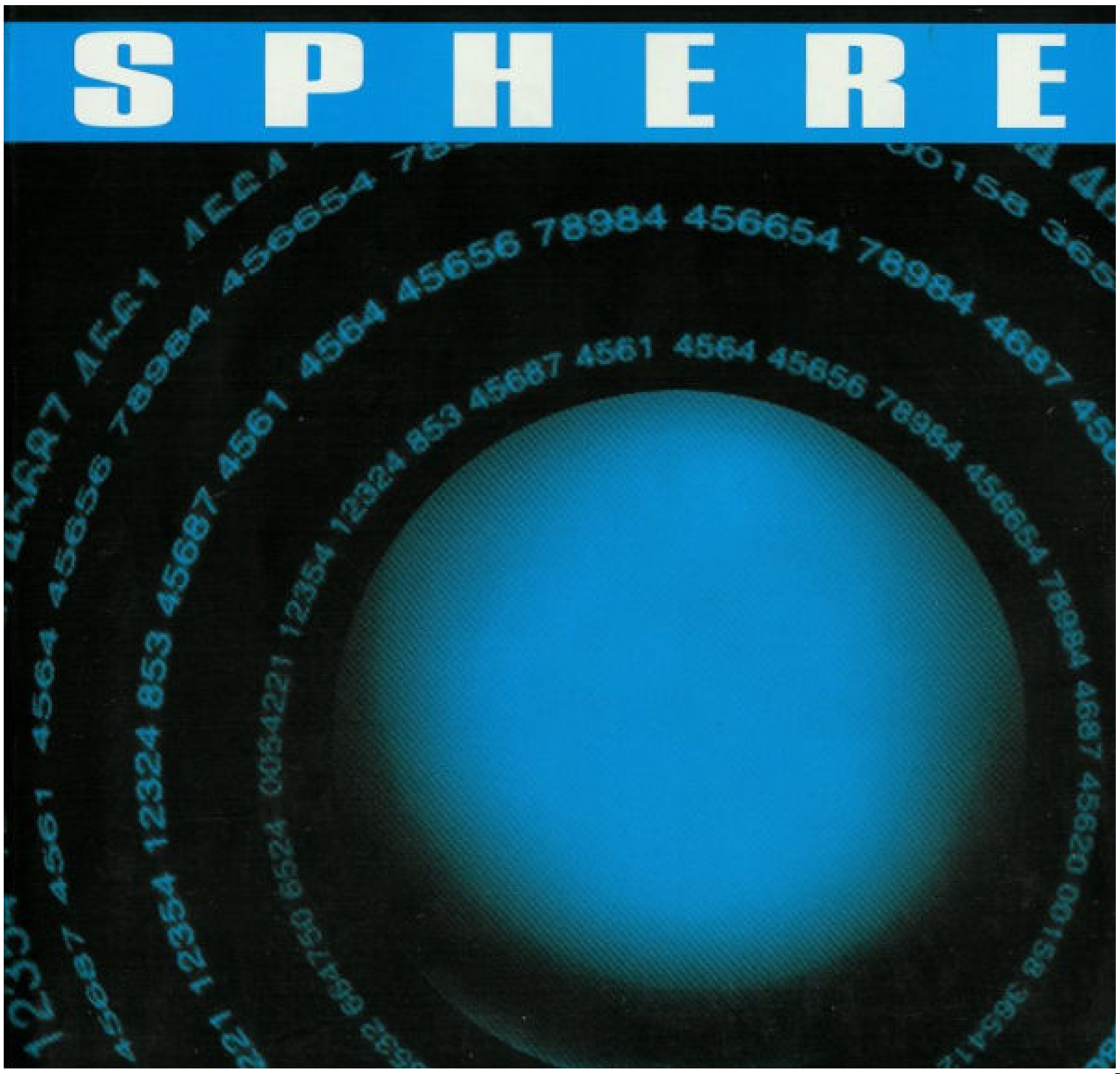

...and so it began...

20 years ago, I wrote my first substantial piece of "forensic" techology documentation. There's a very long backstory to that, which is something I'll explore in another document, but the short version is this: My two best friends from University and I put in a bid on a huge lot of then-obsolete technology from our old school, Concordia University...and we won!

I think our bid was for $72.00, which was way cheaper than the school would have had to pay a garbage disposal company to come and haul the stuff away. Anyway, the upshot of our victory was that it put one of the computer systems we had all earned our degrees on in our hands, along with about a ton (literally) of wires, cables, adaptors and other bits and pieces of mysterious and compelling computer and telephone technology.

When it was in service, this machine was called **`rigel.concordia.ca`**. While at school, my friends and I had spent thousands of hours working on `rigel`. The staff at Concordia, treated `rigel` like the precious resource it truly was. This was still back in the days of the "high priests" of computing, when computing departments were still somewhat remote, and the people who worked there even remoter, often only dealt with through a pane of glass, or a slot in a wall.

Anyway, I was somehow amazed that we now owned `rigel`, a "real" computer, and I wanted to put it in service at our company The Imperators Group. But there was a problem; `rigel` arrived DOA.

This is the story of how I got it working again, which was a kind of homage to my alma mater and a big **thank you** for providing the amazing machine that we got to use every day.

When we bought it `rigel` was, even then, horribly obsolete (1985 - 1987 tech). The hardware was based on a Sun Microsystems 4/380 platform, which was actually an pgraded Sun Microsystems 4/280. Our University had tried to keep it alive and in service for as long as possible, and they had achieved an incredible **ten year service history**. But when it suddenly stopped working one day, they decided it was time to go.

Original Configuration (4/280)

```
Model:      4/280
Codename:   Sunrise
CPU Board:  Sun 4200
CPU:        Fujitsu SF9010 IU
            Weitek 1164/1165 FPU
CPU Mhz:    16.67 MHz
RAM (max)   128 MB
Chassis:    12-slot VME (rackmount)
```

Upgraded Configuration (4/380):

```
Model:      4/380
Codename:   Stingray
CPU Board:  Sun 4300
CPU:        Cypress Semiconductor CY7C601
            Texas Instruments 8847 FPU
CPU Mhz:    25 MHz
RAM (max):  224 MB
Chassis:    12-slot VME (rackmount)
```

At the time this document was produced, there was very little help available online. I spent weeks working on the machine in my basement. First I needed to figure out how to take it apart, so I could understand why it didn't even turn on. That repair was pretty fast. But then I needed to figure out how to get it to boot a kernel.

It took a long time, and I learned a lot on the way. Something that compounded my trouble was that the machine lacked a functioning disk drive, and the tapes that arrived with it did not work. This turned out to be a false trail that wasted a **lot** of time.

Breaking convention with our standing rule of naming our computers after a conqueror, I named the machine `sphere`, after the book by Michael Crichton, which deals with the exploration of an alien spacecraft, and how that experience ignited the imagination, and not always in a good way, of the people involved.

`sphere` (the computer) was quite a trippy experience for me. Up to that point, I had only dealt with PC technology, Sun Microsystems hardware was completely alien, hence the name.

Here's what sphere looked like. I am sorry for the small size of the image, these machines are now found only in computer museums or scrap heaps. I do not have photographs from that era, which was way before the era of camera-enabled smartphones or even widely available digital cameras.

Here's my saga of getting `sphere` working, literary warts and all.

It is formatted for the Linux console. 80 characters, fixed width.

An eyes-friendly PDF version of this document is available.

```
--------------------------------------------------------------------------------
TITLE:
--------------------------------------------------------------------------------

HOWTO:  Netboot a Sun 4/380 on NetBSD via Linux

--------------------------------------------------------------------------------
Copyright
--------------------------------------------------------------------------------

This document is Copyright (c) 1997, Graham Leach.

--------------------------------------------------------------------------------
Disclaimer:
--------------------------------------------------------------------------------

I have made every effort to guarantee that the information within this
document is correct.  However, errors do creep into documentation and the
written word is subject to interpretation.  Therefore, this document makes no
claims, express or implied, as to any relevance to any particular purpose or
pursuit within the realm of human behaviour.

Should you choose to follow the information contained within this document
you do so in the full knowledge that it is at your own risk.  By continuing
to read this document you acknowledge that without proper precautions and/or
corroboration of the facts proposed herein it is within the realm of
possibility that the possible results of this document may be anything,
```

including but not limited to: Your machine going up in smoke, your spouse leaving you, your dog biting you and your machine not netbooting.

By continuing to read this information you agree to release me from any and all possible legal pursuits, of any kind - imagined or real.

Index

Topic 01. Intent

This document details steps required to produce the following results:

"To enable a Sun computer to boot the NetBSD Operating System over its Ethernet interface via services provided by Linux host(s)."

Topic 02. Online Resources

This is an online document. Please view the latest version, along with supporting information, at:

~~http://www.imperators.com/sun/netboot/index.html~~
http://graham-leach.com/MEDIA/HOWTO-Netboot-a-Sun-4-380-on-NetBSD-via-Linux.pdf

Here are the other necessary resources required:

Component	Resource	Note
ARP server	Linux Kernel	built-in
RARP server	Linux Kernel	must be enabled
TFTP server	Linux inetd daemon	must be enabled
BOOTPARAMD server	Xkernel package	Must be installed
NFS server	Linux nfsd daemon	built-in
	Linux mountd daemon	built-in
TAR archive utility	GNU TAR	must be installed
NetBSD OS	NetBSD base.tgz	must be installed

Topic 03. Resources Locations:

The packages necessary to enable these resources are available from:

1) Linux Xkernel

    ftp://sol.ctr.columbia.edu/Xkernel

2) Tar

    ftp://ftp.fsf.org

3) NetBSD base.tgz

    ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-1.3.2/sparc/binary/sets/base.tgz

4) Linux Operating System

    http://www.linux.org

Topic 04. Do resources already exist that concern netbooting Suns?

YES.

There are many websites surrounding the topic of netbooting Suns. Sadly, they almost exclusively concern netbooting Sun 3/50 computers as Linux X-Terminals.

NOTE: A Sun 3/50 is not a very powerful computer. They have little Motorola 68020 processors. They only support a maximum of 16Mb of RAM. Not very exciting. But they do make very nice X-Terminals. Why? They have lovely big 19" displays, the "right" keyboard and they are CHEAP.

ftp://sol.ctr.columbia.edu/Xkernel contains a bunch of interesting and useful information on netbooting Sun 3/50s. The object of the Xkernel project is to enable enterprising computer-type people to recycle Sun 3/50 machines as Linux X-Terminal clients.

While the Xkernel effort doesn't exactly match what this document is trying to outline, the resources collected by the maintainers turned out to be very useful to the writer of this HOWTO.

As it turns out, many of the tools presented here are those also used to boot Sun 3/50 Xkernel machines. When it comes to netbooting, there is no big difference between Sun 3 and Sun 4 series computer. The difference is WHAT they boot, now HOW.

Topic 05. What is required to get a Sun to boot over the network?

Before I get into the specifics of how to get a particular Sun to attempt a netboot, let's outline the conditions that must be satisfied for any successful Sun netboot:

1) You must have a Sun computer.
2) The Sun network interface must be specified as its boot device.
3) The Sun network interface must be functioning properly.
4) The Sun network interface must be connected to the network.
5) You must have a Linux computer.
6) Linux must be functioning correctly.
7) Linux must be connected physically to the network.
8) Linux networking must be configured properly.
9) Linux must have a properly configured RARP server.
10) Linux box must have a properly configured TFTP server.
11) Linux box must have a properly configured bootparams server.
12) Linux must have a properly configured NFS server.
13) Linux must have the required boot file(s) in place for Sun.
14) Linux must have the required Operating System in place for Sun.

Topic 07. A Sample Sun - Hardware Configuration

Here is the hardware profile of my Sun:

Sun 4/380
Ethernet Card
32Mb RAM
A Wyse Terminal attached to serial port "A"

Topic 08. A Sun Boot Sequence

Here is what I saw when I booted my Sun on for the first time:

```
________________________________________________________________________
        FDX         8:01a    1-  1
------------------------------------------------------------------------
EEPROM:  Using RS232 Port A as input device
Selftest Completed.

Sun SPARCsystem 3XX
ROM Rev 3.0.3, 32MB memory installed, Serial #15234.
Ethernet address 8:0:20:8:FA:51, Host ID 23003B82

Testing 1 Megabytes of Memory ... Completed.

Auto-boot in progress...

EEPROM boot device... sd(0,0,0)

sd: Device not found.
>

________________________________________________________________________
```

Allow me to explain what is going on:

1) The first thing the Sun does when it boots is look for a means by which it can interact with the operator. Once it finds a valid input/output device (this is called the console), it displays that information on the console. This results in the message "EEPROM: Using RS232 Port A as input device" appearing on the display.
2) The computer then performs a Power On Self Test (POST). Successful completion of this test is indicated by the message: "Selftest Completed."
3) The computer then attempts to load the Secondary Stage Boot Loader from its default boot device. In this case, the default device is not the Ethernet interface. It is a SCSI disk. The fact that it is looking for a SCSI disk is indicated by the message: "b sd(0,0,0)".
4) The SCSI disk was not found, creating a fault condition. This is indicated by the error message "sd: Device not found."
5) A fault condition exists. At this point the Sun boot sequence terminates and the machine requests operator intervention. This is indicated by the MONITOR prompt ">".

Obviously the first thing that needed to be changed was the default boot device. I wanted my Sun to boot from the Ethernet interface.

Presumably you want to do this too. Configuring a Sun to automatically boot from the Ethernet interface requires that its internal settings be changed. Sun computers store settings such as that of their default boot device in a special type of non-volatile memory called Electronically Erasable and Programmable Read Only Memory, or EEPROM.

Like all Read Only Memory (ROM), EEPROM memory can be programmed. Like all ROM, EEPROM memory retains its values when the host in which it resides is turned off. What distinguishes EEPROM from other ROM memory is that it can be re-programmed. Most ROM technology is a one-shot affair and can only be programmed once. If anything changes the ROM chip has to be physically changed.

In the Sun, EEPROM settings can be changed at the ">" prompt of the MONITOR program.

Topic 09. What is This > Prompt? What's a MONITOR Program?

The ">" prompt is the MONITOR program in action.

Just about every computer ships with some kind of MONITOR program, although some implementations are more complete than others. In general, think of the MONITOR as a bare-bones OS with small set of tasks:

1) Perform a hardware system check.
2) Enable interaction between the computer and the operator.
3) Load an Operating System from the default boot device.

The Sun MONITOR program is very cryptic and utilitarian. That is because its sole purpose is to prepare the computer for the "real" Operating System - to whom it relenquishes control immediately upon load. When everything goes right you don't even notice that the MONITOR is there.

Topic 10. What devices can a Sun 4/380 boot from?

There are a lot of boot devices available within the MONITOR program. These boot devices can be listed at the MONITOR prompt with the "b?" command:

Here is what my Sun displays when I enter that command:

```
>b?
Boot syntax: b [!][dev(ctlr,unit,part)] name [options]
Possible boot devices:
  id:  PANTHER (ipi) controller
  xd:  Xylogics 7053 disk
  xy:  Xylogics 440/450 disk
  sd:  SCSI disk
  le:  Sun/Lance Ethernet
  gn:  Sun Generic Network
  xt:  Xylogics 472 Tape
  st:  SCSI tape
  sr:  SCSI CDROM
>
```

As you can see, a Sun can boot from tape, disk, CD-ROM and two different

Ethernet devices. That's a pretty astonishing number of choices.

There are two ways to boot a Sun. One is to specify the device as the default boot device in the EEPROM, the other is to specify the interface at the MONITOR prompt. Let's look at these choices in turn:

Topic XX. How can I boot a Sun on the Lance Ethernet Interface?

One of the boot options listed is "le (Sun/Lance Ethernet)". Using the syntax notation that is displayed at the beginning of the list of boot devices to phrase the boot command, use:

```
b le(0,0,0)
```

To instruct your Sun to boot the Ethernet interface. If your EEPROM has been programmed to boot from the first device on the first controller, you can omit the 0,0,0 of the boot command and just enter:

```
b le()
```

And it will boot just as if you had typed in the full command.

Topic 11. How do I automatically boot the Ethernet Interface?

Automating the boot process so that the Lance Ethernet interface is used as the default boot device when time the Sun is started is accomplished by altering two values in the EEPROM, at locations 0x019 and 0x01A. Here is a listing of the steps I performed to change my default boot device from the SCSI disk to the Lance Ethernet device:

```
>q 119
019:73 ?? 6C
01A:64 ?? 65
01B:00 ?? q
>
```

NOTE: The EEPROM memory location and its value is always displayed when you enter the EEPROM editor by pressing "q" at the MONITOR prompt. Pressing without typing any values at the ?? prompt allows you to view the EEPROM value of that location WITHOUT CHANGING it.

Had I not entered new values for locations 0x019 and 0x01A they would have remained at values 0x73 and 0x64 - SCSI disk.

Refer to the document "Sun3 BOOTROM Page" in your favorite www search index to find more interesting values that concern Sun EEPROM (re)programming. Much of this section has been derived from that document.

NOTE: The "Sun3 BOOTROM Page" is c Heiko Krupp, 1996.

Once the EEPROM has been altered you can restart the Sun by entering k2 at the MONITOR prompt. This requests a "cold reboot" from the MONITOR program. You no longer need to power cycle your Sun, unless of course that is what you want to do.

```
>k2
```

If you do choose to power cycle your machine do it like this:

1) Turn the machine off.
2) Wait 10 seconds or so.
3) Turn the machine on.

Either way, when you boot the Ethernet interface will be the default boot device and you will see the first step of your task completed.

The first step of a Sun network boot looks like this:

```
_______________________________________________________________________
FDX         8:01a    1-  1
-----------------------------------------------------------------------
EEPROM:  Using RS232 Port A as input device
Selftest Completed.

Sun SPARCsystem 3XX
ROM Rev 3.0.3, 32MB memory installed, Serial #15234.
Ethernet address 8:0:20:8:FA:51, Host ID 23003B82

Testing 1 Megabytes of Memory ... Completed.

Auto-boot in progress...

EEPROM boot device... le(0,0,0)
Requesting Internet address for 8:0:20:8:FA:51
/
_______________________________________________________________________
```

Let me describe what is happening. The Sun is:

1) Loading the MONITOR program from EEPROM
2) Running self-checks to guarantee machine "ready" state
3) Executing the Primary Stage Boot Loader (PSBL) Program
4) Asking for its IP address via a RARP request

Topic 12. What is a PSBL? Why do I need one?

Primary Stage Boot Loaders exists to help computers load their Operating System. They are also known as "bootstrap routines". They are usually implemented in non-volatile memory (in our case, EEPROM). Their sole purpose is to locate and load the Secondary Stage Boot Loader and then execute it.

Topic 13. Why TWO boot loaders? Why isn't one enough?

The PSBL is not very flexible - don't forget that it has been burned into ROM. It is limited to the devices and routines that existed when it was created. It is limited in terms of space. ROM and EEPROM memory are expensive and small in terms of memory capacity. To address these limits someone smart came up with the Secondary Stage Boot Loader (SSBL) concept.

A SSBL is implemented in software. It can be re-written to take advantage of new algorithms and devices. It can be used to extend the life and usefulness of the PSBL by expanding upon its basic boot process. The best feature of the SSBL is that it can be maintained through software because it IS software. It can be distributed and installed like software.

NOTE: This solved an old problem. In older times, installing a more flexible bootstrap routine required operators to re-program the entire thing. Take it from someone who has keyed in a fairly large bootstrap routine using toggle switches just to add a few more bits of information, SSBL programs are a wonderful idea!

Topic 14. What's RARP?

RARP stands for Reverse Address Resolution Protocol. I will discuss what it is once we have guaranteed that the Ethernet interface on the Sun is functioning correctly. Without functioning hardware a discussion of RARP is premature.

Topic 15. How do ensure that the network interface is functioning?

Chances are your network interface is functioning correctly until proven otherwise. I say this because you may not have any tools to troubleshoot the Ethernet card on your Sun aside from the status LED's on the CPU card itself.

The only message that the Sun can give you with these LED's is that the box is functioning correctly on a hardware level. This is indicated by the fifth LED in the series flashing. This means the system is in a NORMAL state and that no hardware errors have been detected by the self-check that the Sun normally executes on power-on.

Sun ethernet interfaces don't have status LED's like in the Intel world. The only way you can GUARANTEE that the network interface is working right is to connect it to a network and use another computer to verify that the Sun is issuing packets onto the network media. That comes up later.

Topic 16. My Sun only has an AUI port. What about a Thin/UTP network?

You may not be able to connect your Sun to your network physically if it has an old Ethernet connector and your network is of a more recent variety.

I was faced with this problem. My Sun featured two Ethernet interfaces, but both were of the AUI variety (a fifteen-pin "D" type connector). This connector is indicative that the machine came from a Thick-Ethernet environment.

My home Ethernet environment is either Thin-Ethernet (Coaxial-BNC) or Twisted-Pair-Ethernet (UTP-RJ45). To connect my Sun to the network I had to obtain a MEDIA CONVERTER. My Media Converter came from a local computer salvage store.

NOTE: If you're not lucky like me and have a salvage store nearby, you can obtain a new unit from Black Box at http://www.blackbox.com or hit the Internet and look around for one. Keywords to use are AUI, Ethernet, Media Converter, UTP, BNC.

Let me describe to you what a media converter looks like, and what it does. My CABLETRON Media Converter is a square box about the size of a Big Mac. It has a fifteen-pin AUI connector on one side and a coaxial (BNC) connector on another. It has a line of LED's on its face.

Here is what is written on it:

```
-----------------------------------------
ST-5XX with LANVIEW(tm)
ETHERNET/IEEE 802.3 TRANCEIVER UNIT (MAU)
FCC ID: F4T4K9-ST-5XX
CABLETRON/DIV OF THE COLLINGSWORTH CORP.
CERTIFIED TO COMPLY WITH CLASS B LIMITS,
PART 15 OF FCC RULES.
SEE INSTRUCTIONS IS INTERFERENCE TO RADIO
IS SUSPECTED.
-----------------------------------------
POWER REQUIRED: DC 10-15V, 3XXmA MAX
NEC 725 2(b) U.L. 910
-----------------------------------------
SQE IS USER SELECTABLE SEE USER MANUAL
```

There is writing beside the LED's that indicate the following:

```
PWR (green)
SQE (unlit)
XMT (flashing green when operating)
RCV (flashing amber when operating)
CP  (red)
----------------------------------------------------------------------
```

Installation of the CABLETRON Media Converter was simple.

1) I turned the Sun off.
2) I connected the BNC connector to the Thin-Ethernet network.
3) I connected the AUI connector to the Thick-Ethernet network.
4) I turned the Sun on.

When I powered on the Sun, the power light on the CABLETRON lit up. It glowed a steady green. The CP light also came on. It glowed a steady red. When the Sun tries to boot the Ethernet interface the XMT and RCV LED's flash.

To me the flashing of the LED's showed that the unit was functioning. It also showed that the Ethernet card of the Sun was operating properly and signals were being issued through the Media Converter to the network. But this was just a guess. The only thing I knew for sure was that the CABLETRON unit was detecting and sending Ethernet frames on the network media.

Topic 17. How do I verify that RARP requests are being broadcast?

RARP packets have a signature - the value 0x8035 is in the Ethernet Type field of RARP packets. To GUARANTEE that RARP traffic is flowing on your Ethernet network from your Sun you will need to install a "packet sniffer" on the Linux box to look for packets with that signature.

There are quite a few "packet sniffer" programs available on the Internet. What these packages do is capture ALL traffic flowing over the Ethernet Interface of the machine they are installed on, regardless of who the packet is addressed to. This is quite unlike the normal Ethernet mode where traffic not addressed for that particular card is ignored.

This special mode of the Ethernet interface is called "promiscuous mode". Running a card in this mode represents a VERY big security hazard. Please read the following:

SECURITY ALERT

Do NOT allow users access to packages like sniffit, tcpdump, etherpeek or any other Ethernet sniffer programs. Firstly, these programs run suid root. This is because they must access the Ethernet device in "raw" mode.

If the user manages somehow to "break" the sniffer program and go to a shell, they do so as _root_. Secondly, these programs allow users to monitor network traffic. It allows would-be hackers to reconstruct unencrypted traffic (like telnet passwords and http traffic). They allow users to and "eavesdrop" on all other sessions, even those of root!

That said, here is how to use to detect the RARP broadcast packets being issued by your Sun.

(17)-----> This is the perfect place to put a "packet sniffer" section.

Now that we have verified that RARP requests are indeed flowing over the Ethernet network, the next logical step is to configure a RARP server to be able to answer. It's time to start discussing the Addressing Protocols in the Linux (Unix) world.

Topic XX. ARP, RARP and /etc/hosts. What's the connection?

The connection between them is that all three are implicated in locating, naming and accessing hosts on the network. Most of the time when people think of resolving names in the TCP/IP world they immediately think of DNS.

DNS didn't always exist. In the old days of the Internet there was no need for complex name services like DNS. Every computer in the world (don't forget that at one point you could count them on your fingers) just looked at a local file called /etc/hosts if it wanted to resolve the IP address of another machine.

The file /etc/hosts held entries for all of the computers in the world, their IP address, and any "nicknames" that they may have. The /etc/hosts file was maintained by people at Stanford University and distributed from that central place to all the hosts that needed it.

(18)-----> VERIFY where uucp came from and where /hosts was maintained

This system worked for some time, but failed to scale beyond a certain number of hosts. Soon it was time to re-think the solution. DNS is the flexible, powerful answer to that problem. It solved all of the issues faced by the centrally distributed /etc/hosts approach.

DNS is complicated. It is relies on full network connectivity. It doesn't work well during the initial phase of a computer (re)boot when network interfaces may be unavailable. As a result, many machines who use DNS once they have achieved FULL network connectivity still use entries in the /etc/hosts file at boot time to get things going. This explains why the /etc/hosts is still in use even though DNS is now the primary tool for Fully Qualified Domain Name (FQDN) to Internet Protocol (IP) address translation.

The /etc/hosts file also gets used for other things. One of the things that it is used for is to support the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP). For RARP to work properly, the IP address of a RARP client must exist in the /etc/hosts file of the RARP server. This is because RARP needs an "authority" to consult. The other thing that RARP needs is a value to return to the RARP client - an IP address.

You must choose an IP address for your netbooting machine.

For my Sun I chose 192.168.0.150

To enable RARP to help the Sun boot, I had to edit the /etc/hosts file on my Linux box. I added an entry for my Sun, as follows:

```
# hosts         This file describes a number of hostname-to-address
#               mappings for the TCP/IP subsystem.  It is mostly
#               used at boot time, when no name servers are running.
#               On small systems, this file can be used instead of a
#               "named" name server.  Just add the names, addresses
#               and any aliases to this file...
#
# By the way, Arnt Gulbrandsen  says that 127.0.0.1
# should NEVER be named with the name of the machine.  It causes
# problems for some (stupid) programs, irc and reputedly talk. :^)
#

# For loopbacking.
127.0.0.1 localhost
192.168.0.150  sphere         sphere.network.domain
192.168.0.1    linux          linux.network.domain
# End of hosts.
```

Let me explain the entries in this file:

The first entry is for the loopback device. The loopback device is a very special device. It is used by various kernel processes. It is used to help the machine refer it itself. DNS uses it. It's special. Leave it alone.

The second entry is for the Sun machine. The first value on the line is the IP address. The second value is the name by which I wish the computer to be known (The name sphere refers to a movie where a group of people attempt to explore and make sense of an alien artifact - exactly what I considered this computer when I got it!). The third value is a fuller name (FQDN) of the computer. As a result the Linux computer would resolve either "sphere" or "sphere.network.domain" to the IP address 192.168.0.150.

The third entry is for the Linux machine. The entry is there so Linux knows its own IP address even when named (DNS) is not running. It is also important for an upcoming step that involves NFS and bootparamd. Due to the limitations of some of the networking protocols involved in netbooting Suns, the short name of the computer is sometimes the only value that can be used - because of the size of buffers being allocated during the netboot process, the FQDN of a computer can sometimes be too long.

Topic 18. What do ARP and RARP stand for? What do they do?

Address Resolution Protocol (ARP) is a pretty important protocol. It is used for TCP/IP packet delivery. It does this by performing an important service: IP to Ethernet address translation.

Let us look at this from the point of a scenario. Say a machine is acting as a router, or "gateway" for some network. It receives an IP packet with an address in the range of the network for which it is responsible, but not addressed to itself. To deliver the IP packet, it consults its ARP cache and looks for the MAC address that corresponds to that IP address.

If it doesn't find the MAC address within its internal tables, it sends out an ARP query. The ARP query effectively says "Hey! Whatever machine has the IP address xxx.xxx.xxx.xxx please tell me their MAC address, I have something for you...". The machine for which the packet is destined says: "If you're looking for IP address xxx.xxx.xxx.xxx you want me. My MAC address is XX:XX:XX:XX:XX:XX".

The gateway machine then formats a frame based on the media type that the destination machine is on, puts the MAC address in the appropriate spot and releases the frame onto the media. It also caches the ARP reply in case more packets for that machine come in the near future.

NOTE: That's one of the reasons why TCP/IP is portable. While Ethernet addresses are media specific, IP addresses are not. So TCP/IP traffic can flow over such media as Token-Ring, Arcnet, FDDI and bunches of other exotic networking technologies without worrying about media-specific address limitations. That's one of the most powerful features of TCP/IP in a nutshell.

Here's a web citation that discusses ARP and RARP:

> http://www.cs.arizona.edu/classes/cs525/manual/node73.html
>
> "D. Plummer. An Ethernet Address Resolution Protocol. Request for Comments 826, USC Information Sciences Institute, Marina del Ray, Calif., Nov. 1982.
>
> ARP translates IP addresses into Ethernet addresses, and vice versa (i.e., it also implements RARP)."

Reverse Address Resolution Protocol, or RARP, does the opposite of ARP. RARP broadcasting enables a machine to request its IP address by telling the RARP server its Ethernet Address.

RARP is a less commonly used facility, especially in a Linux context. It is used primarily to enable diskless workstations (like Sun 3/50 Xterminals) to netboot. As a result it is not ordinarily compiled into a Linux kernel - 99% of the time it would add code that would never be used. In the interests of efficiency and kernels that fit within 1.44Mb of space it is usually left out.

Let us make clear what is going on when sphere issues a RARP broadcast, and why RARP is so important to the netboot process in general:

When ANY computer boots it doesn't know much about itself. That is, other than what is in its non-volatile memory (EEPROM or ROM). Aside from that collection of information, there may be other values in the machine in expansion equipment (like network cards) that has been "burned in" during its manufacture. It's pretty bare-bones.

NOTE: One of the things "burned in" to a network enabled machine is the Ethernet Address of its Ethernet Interface. The Ethernet Address is burned into the Ethernet circuitry during its manufacture.

Furthermore, Ethernet addresses are guaranteed to be unique because their assignment is governed by a central authority. Ethernet card manufacturers are given exclusive classes of Ethernet addresses to "burn" into their devices. This practically eliminates the possibility of a duplicate Ethernet Address existing in two different Ethernet cards at the same time (barring human intervention or error).

A protocol was developed to enable network booting of diskless computers based upon using the Ethernet Addresses of that machine. Why the Ethernet Address? The designer(s) of the protocol were very confident that no confusion would result over the identity of the netbooting client if Ethernet Addresses were used. They built a framework to support netbooting based upon the following assumptions:

Another system on the network (say, a server) can respond to a boot request (RARP) by using the broadcast Ethernet address as a "key" to identify resources specific to the machine issuing the request. The server could then present those resources to that machine over the Ethernet medium because one of the values embedded within the broadcast is the destination address. This mechanism allows one computer to help another boot through the use of a single 48-bit number. Pretty neat.

Let's use sphere to illustrate RARP in action:

sphere boots and issues the following information over its Ethernet interface:

"My Ethernet address is 8:0:20:8:FA:51. What is my IP address?"

The RARP server consults its internal RARP cache and responds:

"Ethernet address 8:0:20:8:FA:51 maps to IP address 192.168.0.150"

The netbooting computer accepts that information. Armed with its IP address, it can now move on to the next stage of its boot process. It uses the IP address it was given to identify itself from that point onward.

Now that the technology has been explained, let's get it working!

Topic 19. How do I install ARP on a Linux machine? How do I configure it?

ARP is one of the rare resources mentioned in this paper that comes without any thought or effort whatsoever. ARP is built in to Linux and it takes care of itself.

Topic 20. What does it look like when ARP is NOT working properly?

Your packets won't get delivered properly. Your network will be dead. Ping and TELNET will not work. In fact, nothing will work.

Topic 21. How do I know that ARP is working properly?

Your network will function correctly. Packets will be delivered. Ping and TELNET will work.

Topic 22. How do I install RARP on a Linux machine?

RARP is trickier to get going than ARP. RARP has to be enabled. In my case (Slackware) RARP is an optional component that can be compiled into the kernel, or a module that must be compiled so it can be inserted into a running kernel.

You have a choice to make. You can either build a kernel that has RARP built-in, or you can choose to insert the RARP module into your running kernel and see what happens.

Topic 23. Building a RARP-enabled kernel

There are two ways to do a kernel build. Both involve altering the /usr/src/linux/.config file:

1) Run the /usr/src/linux/menuconfig program. This script acts as a front end tool and uses your responses to (re)write the /usr/src/linux/.config file.
2) Edit the /user/src/linux/.config file directly.

Here are the two methods:

(1)
```
:/#cd /usr/src/linux
:/usr/src/linux#make menuconfig
Networking options
< > IP: Reverse ARP
```

You must say "Y" in the appropriate place. An asterisk must appear between the GT/LT "<*>" for this option be on.

(2)
```
:/#cd /usr/src/linux
:/usr/src/linux#joe .config
Change:
"CONFIG_INET_RARP is unset"    =>      "CONFIG_INET_RARP=y"
```

Topic 24. How do I Compile a Linux Kernel?

Telling you how to compile a kernel is beyond the scope of this document. Look at the Kernel HOWTO for a very complete discussion of how to compile and install a new kernel on your Linux computer. On my system I would read it with the following command:

```
zless /usr/doc/faq/howto/Kernel-HOWTO.gz
```

Topic 25. How do I Install a Linux Kernel?

After building the new RARP-enabled kernel, It must be installed using LILO. The system must then be rebooted. LILO is another topic that is beyond the scope of this document. Please look at the LILO HOWTO. On my system I would read it with the following command:

```
zless /usr/doc/faq/howto/mini/LILO.gz
```

Topic 26. FAQ this, FAQ that! Too much reading! Isn't There a Faster Way?

OK. Here's something for the daredevils!

```
---------------------------------------------------------
FOR THE IMPATIENT!  NO GUARANTEES!  I AM NOT RESPONSIBLE!
---------------------------------------------------------
******I would have a boot disk ready if I were you!******
---------------------------------------------------------
```

```
:/#cd /usr/src/linux
:/#usr/src/linux#make dep ; make clean ; make zImage ; \
make modules ; make modules_install

:/#usr/src/linux#cp ./arch/i386/boot/zImage /vmlinuz
:/#usr/src/linux#lilo
:/#usr/src/linux#shutdown -r now
 ?
```

Topic 27. Using Modules to RARP-Enable a kernel

If you're lucky, you can enable RARP simply by issuing the following command(s):

```
:/#insmod rarp
:/#lsmod
Module:     #pages:     Used by:
rarp            1              0
iBCS           25              0
ds              2              0
pcmcia_core     7       [ds]   0
ppp             5              0
slip            2              0
slhc            2       [ppp slip]      0
lp              2              0

:/#
```

Note that on line one (1) of the output of the "lsmod" command, rarp has been loaded as a module and is available.

NOTE: Once you verify that the command is correct, don't forget to add it to your /etc/rc.d/rc.local file. If you forget this important step, your Linux machine will "forget" that your netbooting computer exists!

This will cause very strange netboot behavior after the Linux

        server has been (re)started. Suddenly your netbooting Sun will
        not work! Save yourself the grief and do it now.

Here's what my /etc/rc.d/rc.local looks like:

```
#
#
#
#
```

--? INSERT APPROPRIATE TEXT HERE

Topic 28. How do I configure RARP?

The RARP cache has to be "primed".

For sphere this is done by executing the following:

```
:/#/sbin/rarp -s sphere 8:0:20:8:FA:51
```

The RARP cache can be dumped to the screen (and verified) with the
following command:

```
:/#/sbin/rarp -a
IP address       HW type             HW address
192.168.0.150    10Mbps Ethernet     08:00:20:08:fa:51
```

As you can see, the IP address 192.168.0.150 has been linked with the
Ethernet address 08:00:20:08:fa:51. The connection between the Ethernet
address 08002008FA51, the name "sphere" and the IP address 192.168.0.150
has been successfully established!

How do I know? The name "sphere" doesn't appear in the list that is
produced by RARP. While the name "sphere" is used in the "priming" step
to help form the connection, its IP address appears in its stead when
RARP is asked to produce its table. This is verification that the RARP
entry worked. The RARP entry forms a between the Ethernet address, the
name of the netbooting machine in the /etc/hosts file, and the IP
address that also appears in that file.

NOTE: Once you verify that the command is correct and the information in
        the RARP cache is right, don't forget to add it to your
        /etc/rc.d/rc.local file. If you forget this important step, your
        Linux machine will "forget" that your netbooting computer exists!

        This will cause very strange netboot behavior after the Linux
        server has been (re)started. Suddenly your netbooting Sun will
        not work! Save yourself the grief and do it now.

Here's what my /etc/rc.d/rc.local looks like:

```
# ------------------------------------------------------------------------
# Graham Leach
# 1997-03-01
# To enable netbooting from this machine, RARP must be enabled
# ------------------------------------------------------------------------
echo Installing RARP
insmod RARP
```

```
# -----------------------------------------------------------------------------
# Graham Leach
# 1997-03-01
#
# A RARP entry is necessary for a machine to netboot.
# -----------------------------------------------------------------------------
echo -n "Priming RARP cache for: "
/sbin/rarp -s sphere 08:00:20:08:FA:51
echo "sphere "
```

Topic 29. What does it look like when RARP is NOT working properly?

RARP isn't working properly when an IP address isn't handed to a RARP broadcasting machine. Here's what I see when RARP is not working right:

```
________________________________________________________________________
FDX         8:01a    1-  1
------------------------------------------------------------------------
EEPROM:  Using RS232 Port A as input device
Selftest Completed.

Sun SPARCsystem 3XX
ROM Rev 3.0.3, 32MB memory installed, Serial #15234.
Ethernet address 8:0:20:8:FA:51, Host ID 23003B82

Testing 1 Megabytes of Memory ... Completed.

Auto-boot in progress...

EEPROM boot device... le(0,0,0)
Requesting Internet address for 8:0:20:8:FA:51
/
________________________________________________________________________
```

That's where the boot ends. If the machine cannot obtain its IP address from the RARP server, it cannot determine the name of the SSBL that it has to load for the boot to continue on. Full stop.

Topic 30. How do I know that RARP is working properly?

RARP is working properly when your server responds to the RARP request and gives the client computer the IP address as specified in the /etc/hosts file:

```
________________________________________________________________________
FDX         8:01a    1-  1
------------------------------------------------------------------------
EEPROM:  Using RS232 Port A as input device
Selftest Completed.

Sun SPARCsystem 3XX
ROM Rev 3.0.3, 32MB memory installed, Serial #15234.
Ethernet address 8:0:20:8:FA:51, Host ID 23003B82

Testing 1 Megabytes of Memory ... Completed.

Auto-boot in progress...
```

```
EEPROM boot device... le(0,0,0)
Using IP Address 192.168.0.150 = C0A80096
tftp: time-out
tftp: time-out
```
____

Let me explain what is going on. sphere is:

1) Loading the MONITOR program from EEPROM
2) Running self-checks to guarantee machine "ready" state
3) Executing the Primary Stage Boot Loader (PSBL) Program
4) Asking for its IP address via a RARP request
5) Obtaining its IP address from the Linux RARP server
6) Timing out on a call to TFTP

As you can see above, everything is in place in terms of the Address Protocols. The machine called sphere now knows its IP address. The failure at this point is with TFTP a higher level boot protocol than RARP.

Our discussion now moves to TFTP. Up the networking layers we go!

Topic 31. What is TFTP? What does it do?

Trivial File Transfer Protocol is used to transmit files between networked computers. It can be used to boot diskless computers. It is usually automatically started when required by the inetd "SUPERSERVER" daemon.

NOTE: One of the packages required for a successful Sun netboot is Xkernel. It contains a specially modified version of TFTP that transmits files in 1k blocks. This is the block size that Sun computers require. The Xkernel TFTP is different than the TFTP which comes with Linux. It transmits in 8k blocks. Another difference is that the Xkernel TFTP daemon looks for files in a specific place in the file system. Here is a relevant section of the Xkernel documentation that describes this special tftp daemon:

> "This package includes a specially modified tftpd server. To increase security, it prepends usr/export/tftpboot/ to the pathname of all requested files. This helps to remove a bug found in some Linux tftpd's that causes them to refuse requests for files that don't have a complete (from the root) path."
>
> The Xkernel package is copyright:
>
> ---> Get the Xkernel credits together here.

As I stated before, inetd is responsible for starting many services. One of these services is tftp. To have Linux inetd start the "right" TFTP daemon when a TFTP request is received it must know where the tftpd program is. It also must know under which uid to run the tftp program. All of this information is held in the inetd.conf configuration file.

NOTE: TFTP is a service offered by the Unix "SUPERSERVER" - inetd. At

one time there was a HUGE collection of single purpose programs that answered requests for services in Unix. Hundreds of services begat hundreds of little daemon programs. The services were so popular and requested so frequently that someone decided to create a big program that held all of the most popular servers (daemons)
within it. The reasons behind this have to do with systems efficiency and architecture. Basically the superserver helps to reduce context switching, enables a consistent invocation medium and addresses other important issues inherent in a separate implementation strategy.

Topic 32. How do I install TFTP on a Linux computer?

TFTP is another "dormant" Linux feature that has to be turned on. Fortunately, a kernel build is NOT necessary. All that has to be done to enable the TFTP server is a small adjustment to the /etc/inetd.conf file on the Linux machine. The /etc/inetd.conf file is quite large. You will have to look for the string "tftp" to locate the appropriate section. Here's what mine looked like before and after I edited it:

BEFORE

```
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
#
#tftp  dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
```

AFTER

```
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
#
# tftp  dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd
#
#----------------------------------------------------------------------
# Graham Leach (98.09.23)
# Following instructions in /usr/export/README:179
# I have added the following one (1) line(s).
#----------------------------------------------------------------------
tftp dgram udp wait nobody /usr/sbin/tcpd /usr/export/src/tftpd/tftpd
```

Here's an explanation of that line:

tftp	: This is the service name
dgram	: This defines the socket type
udp	: The User Datagram Protocol is used
wait	: A special condition for tftp.
nobody	: Execute this program as user "nobody"
/usr/sbin/tcpd	: Use this program as a "wrapper"
/usr/export/src/tftpd/tftpd	: This is the executable to run

Here is some information that explains the special "wait" condition for tftp in inetd.conf. This excerpt is from man(8):

"Tftpd(8) is an exception; it is a datagram server that establishes pseudo-connections. It must be listed as ``wait'' in order to avoid a race; the server reads the first packet, creates a new socket, and then forks and exits to allow inetd to check for new service requests to spawn new servers."

Once the /etc/inetd.conf file has been adjusted, inetd must be told to reload its configuration file. If this is not done the changes will not take effect until the next reboot.

This can be done with the following command:

```
:/#kill -HUP `ps ax | grep inetd | cut -c 1-6`
```

Now that the Linux server has been configured to respond to TFTP requests, a question remains to be answered. What does sphere want? Certainly, when sphere issues a TFTP request it doesn't just stop at a TFTP server being started on the Linux machine. sphere is asking for something via TFTP. What is it?

Topic 33. What IS this file that TFTP is being asked to provide?

When a netbooting computer issues a TFTP request, it is looking for its Secondary Stage Boot Loader, or SSBL.   Usually this file resides on some sort of local media. In the case of a netbooting computer, TFTP is taking the place of that local mass storage device and providing that service via a network connection.

Let me explain again what a Secondary Stage Boot Loader is. It performs a job very similar to the PSBL - in fact, some would say that it is merely an extension of the PSBL. It certainly has a similar mode of operation - it too exists to load a program, then execute it.

But that is where the similarity ends. There are critical differences: In the case of the PSBL the program being loaded is the SSBL. In the case of the SSBL the program being loaded is an Operating System kernel. There is another important distinction. The SSBL is implemented in software and the PSBL is implemented in firmware (EEPROM).

The SSBL has certain advantages over the PSBL. The SSBL is much more flexible because it can be easily (re)programmed. It can be larger than the PSBL. As a result, it can offer more resources; like additional device drivers, virtual memory and networking (NFS).

NOTE: For an in-depth discussion of boot loader theory, review the LILO documentation. On my machine I would execute the following command to review the literature:

```
:/#less /usr/doc/lilo/README
```

Now we know WHAT the SSBL is. The next step is the find out what FORM the SSBL must be in and WHERE it needed to be located for TFTP to be able to hand it over to sphere when it is asked for.

What is the FORM of the information that sphere is asking for? Quite simple really. sphere is requesting a file whose name matches the IP address that was given to it via the RARP response.

There is one complication - the filename has to be in hexadecimal format, not the familiar "dotted decimal" notation that we (humans) are more used to. Moreover, the characters ".SUN4" be added to the end of the filename. This is because sphere is a Sun 4/xxx machine.

As you may recall, the IP address I chose for sphere is 192.168.0.150.

Using a Windows(tm) calculator (in scientific mode) I translated the IP address to hexadecimal. In my case the numbers translated as follows:

192	decimal =	CO	hexadecimal
168	decimal =	A8	hexadecimal
0	decimal =	00	hexadecimal
150	decimal =	96	hexadecimal

```
192.168.0.150  =      C0A80096
```

Add the suffix .SUN4 and the file becomes C0A80096.SUN4

sphere is asking for the file C0A80096.SUN4 via TFTP. Next question is: Where should C0A80096.SUN4 go so TFTP can find it?

Topic 34. O.K. Now I know what TFTP is looking for. Where do I put it?

When this version of TFTP is called upon to provide a file, it looks in a specific directory. The source to the TFTP code has been altered. This is is to increase security and prevent problems with incomplete file paths from crashing the server.

The directory that TFTP looks in is /usr/export/tftpboot directory.

Here's the scenario: sphere issues a TFTP request asking the Linux TFTP server to provide it with the file C0A80096.SUN4. TFTP is started by inetd and immediately looks for C0A80096.SUN4 in the directory that has been compiled into it. Logically it follows that I had to put something as C0A80096.SUN4 in the /usr/export/tftpboot directory for anything to happen.

My next question was naturally about the file C0A80096.SUN4 - where could I get one?

I hit the Internet. I knew what I was looking for. I needed something akin to the boot section of the boot/root combo for Linux.

I hit upon a promising collection of software. There was an ongoing Linux port to the Sparc platform under the name SparcLinux. The boot loader for the project was called SILO. Unfortunately, it seemed that my Sun hardware (VME bus)was not supported. I bit my lip and moved on ?

Then I hit upon NetBSD.

I'd heard about the FreeBSD effort of course, but I didn't really know very much about it other than the name. I knew that it was another one of the free Unices, that was about it.

What I found was impressive. First of all, my 4/380 was on the Supported Hardware list. Secondly there was a document that described netbooting Sun computers right on their website.

There was not as much supporting information as I was used to with Linux, what was there was excellent. It was also current. The netbooting Sun document I found at that site has greatly influenced the format of this document (Thank You). Here it is:

http://www.netbsd.org/Documentation/network/netboot/index.html

The above document is c 1997 The NetBSD Foundation, Inc.

If you want to know more about the NetBSD effort, here is the resource:

http://www.netbsd.org

By far the most useful thing I found at the NetBSD ftp/website was a file called boot.net. boot.net is a Secondary Stage Boot Loader (SSBL) for NetBSD. But you can call it (taa-daah!) C0A80096.SUN4

I found net.boot at:

ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-1.3.2/sparc/installation/netboot

Setting up the Linux box to hand over the NetBSD SSBL was pretty straightforward. I downloaded the boot.net file via FTP straight into the TFTP /usr/export/tftpboot directory. I then created a symbolic link that pointed to it with the name C0A80096.SUN4.

I did it this way:

```
:/usr/export/tftpboot#ln -s boot.net C0A80096.SUN4
```

I used a link for a few reasons:

1) I wanted to retain the original filename boot.net. Changing the name of the file via "mv" to COA80096.SUN4 not an available option.
2) If I ever get another Sparc that could use this particular boot.net file, I would be able to reference the file with another symbolic link instead of having to copy it. That makes maintenance easier if a new, improved boot.net is ever released and I am forced to replace the one that is there.
3) I did it this way to "future proof" the installation.

The final step here is to make sure that the file permissions on C0A80096.SUN4 were correct so tftp could read the file. If you installed the boot.net file as "root" then TFTP will not be able to open the file unless some permissions are adjusted.

NOTE: This has to be done in such a way as to enable tftp to satisfy the request it receives from sphere without throwing the computer open to the four winds. TFTP is a security problem due to its lack of demand for authentication.

Topic 35. TFTP Security - A Contradiction in Terms

Setting up TFTP properly is really important. TFTP poses a very real security risk because it provides unauthenticated clients access to the filesystem of the server upon which it resides.

While I am sure that you DO want your netbooting clients to be able to access their kernels, you DON'T want to do it in such a way that it leaves you open to damage from persons or processes with less benign intentions. Hackers love facilities like TFTP because they give easy password-free access to the machine. It's like a toehold.

When it came to securing the Linux system, I found that using the following commands really helped:

```
1) chown nobody *
2) su nobody
```

I used command (1) to change ownership of the boot files to nobody. The account nobody is often used in conjuction with providing services to anonymous clients or running processes at a very low security equivalence. That's why the account exists. It has only the most minimal access rights.

Changing ownership of the files to nobody ensured that the Operating System didn't terminate the TFTP process when it tried to access files in trying to provide them to anonymous clients.

NOTE: Don't forget that tftpd is fired up via tcpd when inetd receives a tftp request. When TFTP is fired, it is configured to run under the uid of nobody. This becomes important when hackers try to break TFTP. If they manage somehow to get TFTP to provide them a shell, that shell retains the uid of the owner process - in this case "nobody". It's a lot easier to hack a system if you can obtain the uid "root" than "nobody". To make it safer, TFTP is run as "nobody".

---> VERIFY

I used command (2) when things didn't go right. And not just for this step either. Whenever files weren't being passed correctly and error codes were flying, or what I expected to happen didn't - I would:

```
su nobody
```

And then try to manually navigate to the files from the root directory. If I couldn't get to where I wanted (a la "permission denied"), I would either adjust ownership (or permissions) on files and directories until things started to work.

Let me tell you, a lot of adjustments were necessary to get both the TFTP transfer to work correctly AND satisfy security. I caution you to be patient and thorough. It is worth it in the long run.

```
BEFORE
------
192.168.0.1     linux     linux.network.domain

AFTER
-----
192.168.0.1     linux     linux.network.domain
```

Topic 36. What does it look like when TFTP is NOT working properly?

With TFTP not working, the following appeared when I booted sphere:

```
________________________________________________________________________


>b le()
```

```
Using IP Address 192.168.0.150 = C0A80096
Boot: le(0,0,0)
tftp: time-out
tftp: time-out
-
```

Topic 37. How do I know that TFTP is working properly?

With TFTP working, the following appeared when I booted sphere:

```
>b le()
Using IP Address 192.168.0.150 = C0A80096
Boot: le(0,0,0)
Booting from tftp server at 192.168.0.1 = C0A80002
Downloaded 43784 bytes from tftp server

>> NetBSD/sparc Secondary Boot, Revision 1.7
>> (pk@flambard, Fri Jan  2 00:10:58 MET 1996)
Using IP address: 192.168.0.150
Boot: client IP address: 192.168.0.150
bootparamd: 'whoami' call failed
Can't open network device 'le(0,0,0)'
open: netbsd: Unknown error: code 60
device[le(0,0,0)]:
```

This was GREAT!

Now the client machine (Sun) was:

1) Loading the MONITOR program from EEPROM
2) Running self-checks to guarantee machine "ready" state
3) Executing the Primary Stage Boot Loader (PSBL) Program
4) Asking for its IP address via a RARP request
5) Obtaining its IP address from the Linux RARP server
6) Downloading its SSBL from the Linux TFTP server (43784 bytes)
7) Failing on a 'whoami' call to bootparamd

I initially guessed that "open: netbsd: Unknown error: code 60" really meant "No such file or directory". My reasoning went like this: I was sure bootparamd was not doing its job because of the preceding line: " bootparamd: 'whoami' call failed".

By failing to answer the 'whoami' request, bootparamd was not generating a means for the netbooting client to open something (probably a file) via the network. In essence, the file "open" call failed and the boot halted.

This error bubbled back up and the MONITOR program identified bootparamd as the source of the problem. It seemed that it was time to get bootparamd working.

Topic 38. What is bootparamd?
----

bootparamd is a program designed to respond to netbooting computers to help them load their kernels. It works in conjunction with NFS. It is also used to export file systems to netbooting machines.

Getting a Sun to netboot involves configuring bootparamd to respond to netbooting machines with their name and the location of any other resources they require. It is then up to the netbooting computer to request those via NFS. That is why a discussion of bootparamd MUST also include a discussion of NFS. It also means that the bootparamd and the NFS servers do not necessarily have to reside upon the same computer, but for simplicity they co-exist on the Linux machine for the purpose of this document.

Topic 39. How do I get bootparamd to work on a Linux machine?
----

bootparamd relies on a settings file called bootparams. On my Linux machine this file is located at /usr/export/bootparams.

A bootparams file contains a series of entries, or "stanzas". A stanza is a collection of lines in an identifiable and logical group. A stanza must exist for each netbooting computer.

Here's what my /usr/export/bootparams file looks like:

```
#/usr/export/bootparams
sphere     root=linux:/usr/export/sphere \
           swap=linux:/usr/export/sphere/swap \
           dump=linux:/usr/export/sphere/dump
```

What this entry says is:

1) An entity named "sphere" exists as a client of bootparamd.
2) It has a root filesystem at /usr/export/sphere.
3) Its swap area should be located at /usr/export/sphere/swap.
4) Any core dumps should be placed at /usr/export/sphere/dump.

NOTE: In the Xkernel package the last two settings (swap,dump) are not implemented. The swap area has to be manually implemented on the root file system. Core dumps happen wherever they happen. I figured that putting the settings in now (and creating the directory entries to match) would just "future proof" the installation and reduce work (and frustration) later.

I enabled bootparamd with the following command:

```
:/#/usr/export/src/bootparamd/rpc.bootparamd -f /usr/export/bootparams&
```

NOTE: Once you verify that the command is correct and the information in the ARP cache is right, don't forget to add it to your /etc/rc.d/rc.local file. If you forget this important step, your Linux machine will "forget" that your netbooting computer exists!

This will cause very strange netboot behavior afterLinux server

has been (re)started. Suddenly your netbooting Sun will not work!
Save yourself the grief and do it now.

With bootparamd enabled and running, the netbooting computer will
receive its name:

```
>b le()
Using IP Address 192.168.0.150 = C0A80096
Boot: le(0,0,0)
Booting from tftp server at 192.168.0.1 = C0A80002
Downloaded 43784 bytes from tftp server

>> NetBSD/sparc Secondary Boot, Revision 1.7
>> (pk@flambard, Fri Jan  2 00:10:58 MET 1996)
Using IP address: 192.168.0.150
Boot: client IP address: 192.168.0.150
Boot: client name: sphere
Can't open network device 'le(0,0,0)'
open: netbsd: Unknown error: code 72
device[le(0,0,0)]:
```

Here's what is happening:

1) Loading the MONITOR program from EEPROM
2) Running self-checks to guarantee machine "ready" state
3) Executing the Primary Stage Boot Loader (PSBL) Program
4) Asking for its IP address via a RARP request
5) Obtaining its IP address from the Linux RARP server
6) Downloading its SSBL from the Linux TFTP server (43784 bytes)
7) Obtaining its name from the Linux bootparamd (sphere)
8) Obtaining kernel and file system pointers from bootparamd
9) Failing on a call to NFS.

The next step is to configure NFS to work in harmony with bootparamd so
that it can deliver the resources that the netbooting client requests
from it as a result of a bootparamd response.

Topic 40. What is NFS?

NFS stands for Network File System. It allows computers on a network to
share information by extending the file system concept to network
resources. When Sun invented NFS they wanted to give users access to
information on remote hosts with the same ease as they had on their
local machines. NFS was the answer.

An NFS resource (an export) on one machine (the NFS server) can be
"mounted" upon any point in the directory structure of another machine
(the client) - just like a disk. The resource can then be manipulated
from that point on as if it were a natural part of the directory tree of
the client(with very few exceptions). It's beautiful. Learn about it.
I guarantee you won't be sorry you did.

Topic 41. How does NFS fit together with bootparamd?

NFS and bootparamd fit together in the sense that bootparamd only provides a netbooting computer pointers to information it needs. bootparamd relies on NFS for actual delivery of those resources to the netbooting computer. bootparamd is pretty useless without NFS to back it up.

NOTE: It is the software-based SSBL (boot.net) that knows how to "speak" to NFS, not the hardware-based PSBL that resides within EEPROM in the Sun. The only "network speak" that the Sun PSBL knows is TFTP and RARP. This serves as an example of how SSBL programs can be more powerful and flexible than SSBL programs.

Topic 42. What happens when bootparamd and NFS do their thing?

As I have already explained, NFS has to work in conjunction with bootparamd for a successful netboot to occur.

When the client issues a 'whoami' call, it's more like a "who am I and where am I' call. Take this to mean that what the client wants to know both its identity AND the location of its kernel and file structure. RARP may have told the netbooting computer its IP Address, but it still doesn't know its name. bootparamd provides that information.

The netbooting computer is also looking for software - more specifically, its Operating System. bootparamd tells it where to look by forwarding the netbooting computer values stored in its bootparams file. The client then requests these resources via NFS.

For the netboot to move forward, NFS has to provide these resources (known as exports) to the netbooting (client) computer. NFS must be able to support the directory structure as defined in the /usr/export/bootparams file. The trick between bootparamd and NFS is to get the bootparamd settings file (bootparams) and the NFS settings file (exports) to match.

NOTE: Here's what is happening on my Linux machine from the perspective of the "root" directory (/) of each machine, and where the directories really reside.

```
Linux                  NetBSD
NFS Server directory   NFS Client

usr
|__export
|___root
       |____sphere
           |           sphere:/
           |___root            |___root
           |___swap                   |___swap
           |___usr             |___usr
           |___home            |___home
           |___var             |___var
```

As you can see, the physical organization of the information looks very different to the machine providing the information (the Linux

NFS server) than to the machine receiving the information (the NetBSD NFS client). It's the same information, but the location of the directories in which the information is contained is different.

Another important concept to remember is that the information never actually "leaves" the NFS server. NFS allows the client to think that the information is mounted locally. It is NOT being transmitted like an FTP session would transport information. It is NOT in two places at one time. It just looks that way. Cut off the network to the NFS client and you'll see immediately just how "local" it is!

Topic 43. How do I get bootparamd and NFS to work together?

Setting up NFS and bootparamd to work together is a straightforward process. You have to alter the /etc/exports file and restart the rpc.nfsd and rpc.mountd daemons that compose the service. You have to alter the bootparams file and restart the rpc.bootparamd daemon. The values in the files must match.

Here we go:

This is what my /etc/exports file looks like:

```
#/etc/exports:

/usr/export/root/sphere               sphere(rw,no_root_squash)
```

This entry tells the NFS daemon to allow connections from the machine known as sphere to access the "export" known as sphere.

Furthermore, the files contained in the export "sphere" are located at /usr/export/sphere/root in the NFS server file system. The machine sphere is allowed to both read and write to files in that export.

It also instructs the NFS daemon not to squash the root file system. This is referenced in the man pages for exports, located in man(5):

```
no_root_squash

     Turn off root squashing. This option is mainly useful for
     diskless clients.
```

Topic 44. How do I know that bootparamd is NOT working?

We know it is not working for sure when it reports errors. We've already seen the Code 60 error. Let's go a little deeper into the Code 72 error that results from a properly configured bootparamd and improperly configured NFS:

After I enabled bootparamd on the Linux server, I saw the following:

____

```
>b le()
Using IP Address 192.168.0.150 = C0A80096
Boot: le(0,0,0)
Booting from tftp server at 192.168.0.1 = C0A80002
Downloaded 43784 bytes from tftp server

>> NetBSD/sparc Secondary Boot, Revision 1.7
>> (pk@flambard, Fri Jan  2 00:10:58 MET 1996)
Using IP address: 192.168.0.150
Boot: client IP address: 192.168.0.150
Boot: client name: sphere
Can't open network device 'le(0,0,0)'
open: netbsd: Unknown error: code 72
device[le(0,0,0)]:
```

____

When I first saw this error my guess was that error 72 had something to do with permissions. My reasoning went like this:

1) Error 60 had something to do with file existence.
2) 72 is a higher number than 60.
3) I thought that the first thing a developer would take care of when building a tool was whether or not the file was locatable.
4) Error 72 must concern "higher" functionality than file existence.
5) I was just guessing anyways ?

I felt that file ownership or access permissions had to be at the root of this. A "ls -l" of the directory gave the following permissions for tftpboot:

```
drwx------   3 nobody   root         1024 Sep 26 16:33 tftpboot/
```

Would you agree that directory permissions of 700 could be a problem?

My conclusion was that error 72 meant "Permission denied". when TFTP tried to access the file to was not being allowed access to that directory, only "nobody" could go into that directory, according to the way the permissions were set.

I tested this theory by using an account other than "root" and "nobody". Sure enough, I was not allowed in. To fix it, I had to execute the following command as root:

```
:/#chmod 755 tftpboot
```

This gave the following permissions to that directory:

```
drwxr-xr-x   3 nobody   root         1024 Sep 26 16:33 tftpboot/
```

Then I rebooted the computer one more time. But I did it differently. From sheer experimentation, I had discovered that all I had to do was press CTRL-BREAK and the Sun would drop down into its MONITOR screen and give me a ">" prompt.

Topic 45. How do I know that bootparamd is working?

---> Place a boot with a missing kernel here.

Topic 46. This is all about Sun and Linux. Where does NetBSD fit in?

I know this question sounds a little crazy. But with all this Linux-specific configuration activity, the focus of this document may seem to be a little lost. We ARE going to boot a Sun 4/380 on a NetBSD at the end of this document, I promise. Hang in there. We're almost done.

For the NFS server to have something to export, you have to obtain and install the NetBSD files compiled for the Sparc architecture in the appropriate place on the disk drive of the Linux machine.

To get that boot to happen, NetBSD has to be installed on the Linux server in directory that is NFS-exported as "sphere".

Topic 47. What is NetBSD?

NetBSD is another Unix effort in the spirit of Linux. That means that it is under the care and control of its users. The NetBSD Operating System is free and is open. It comes with its own source code. Users are encouraged to extend the OS as they see fit.

It also comes with no warrantee, express or implied. ?

Here is what the NetBSD people have to say about their own product. This comes direct from the NetBSD homepage, http://www.netbsd.org:

> The NetBSD Project is a collective volunteer effort to produce a freely available and redistributable UNIX-like operating system.
>
> NetBSD runs on a broad range of hardware platforms and is highly portable. It comes with complete source code, and is user-supported.

Topic 48. Where do I get NetBSD?

The definitive source of all NetBSD information, including the location of kernels, utilities and applications for the Operating System is at http://www.netbsd.org.

I found my particular distribution at:

ftp://ftp.netbsd.org/pub/NetBSD/NetBSD-1.3.2/sparc/binary/sets/base.tgz

----
Topic 49. Which NetBSD is the right NetBSD?

The version of NetBSD I chose to install was 1.3.2. As the Operating System matures, I expect that the version number will increment. My advice? Just download the latest "stable" version of the OS and you'll be fine.

I chose to download the one just before that marked as "current". You may choose to live on the dangerous side and download the "current" version of the OS. I must tell you that the "current" version is usually more on the experimental side and is as a result a little less reliable, but often is more fun to run, especially if you consider the cutting edge to be cool.

Topic 50. Is it Dangerous to Install Sparc Executables on an Intel box?

Installing Sparc-executable files on your Linux system won't risk anything. Even if you forget yourself and execute one of them nothing very exciting will happen. You will see the following message from your Intel Linux box:

```
:/#
bash: ./: cannot execute binary file
```

The binary format of Sparc-executables is totally incomprehensible to an Intel-based Linux machine. The message is saying that Linux is guessing that it is a binary file, but cannot make any sense of what is in there. It definitely cannot execute it. So the Linux kernel gives up and issues an error.

Topic 51. How do I install Sparc NetBSD on a Linux system?

It is necessary to extract the netbsd archive base.tgz into an NFS-exported directory on your Linux host. Once this has been done, the client will be able to access and download its NetBSD kernel.

The extraction will also put in place a skeleton file system structure. This file system can be fleshed out with other packages later. Many critical utilities are installed, especially in the /bin, /sbin, /usr/bin directories.

To do this, a recent version of the program known as "tar" is needed.

Topic 52. What is TAR?

TAR is an archiving utility. Archiving utilities are used to collect a bunch of files together into one file. This makes it easier to package them for transport to other computer systems - you only have to keep track of one large file instead of hundreds of little ones. Another advantage of archive utilities is that they preserve the directory structure. Think of the directory structure as the relative position of files to one another from one absolute location. In this case, the absolute location is the "root directory" or "/".

In the old days of Unix, when only tapes existed as the means by which software could be loaded onto a computer, the Utility "Tape ARchive" was created. It was really useful. It's been used ever since, even though the use of tapes to install software has largely become a thing of the past, especially with microcomputers. The last time I really used a tape with a microcomputer was with an Apple II+ cassette port. It was a while ago. ?

Now we use TAR to move information around regardless of the media it resides upon. A tar archive can be transported to disk, tape, over a network, even converted to light and burned into a CD-ROM. They are infinitely flexible and essential to the distribution of packages in the Unix world.

NOTE: tar is useful because it preserves not only the files (information), but also where they are (structure). When used in conjunction with a compression utility, tar can be a very, very powerful tool. It is in use everywhere.

Topic 53. Why do I potentially need a more recent version of TAR?

The structure of the NetBSD base.tgz file contains not just files and their directory structures - it also contains file ownership and permissions information. It is really important to retain file ownership information. This is because of something known as the suid bit. When a file has the suid bit set on. It runs under the privilege level of the owner of the file, not the user executing it.

This can lead to some really horrible security issues. Consider this: The file gets put into the archive owned by the user nobody. The file is extracted by and ownership is set to root. The suid bit of that file happens to be "on".

Now there is a file on the system that when "broken" creates a means by which a person could hack the computer system as root! This is because "broken" suid root programs can give hackers a means of obtaining the privileges of root. This situation is really to be avoided. The new version of tar (1.1.2) retains file ownership information correctly. Older versions do not. For this reason the new version is compulsory for this installation and must be obtained and installed on the Linux system before the contents of the base.tgz file are extracted.

Topic 54. Where do I get TAR 1.1.2?

----

tar version 1.1.2 is available from the Free Software Foundation (FSF):

http://www.fsf.org

Topic 55. How do I install TAR 1.1.2 on my computer?

---> Read the instructions

Topic 56. How do I use TAR 1.1.2 to install NetBSD on Linux?

---> Follow the instructions that come with base.tgz

After you have installed the NetBSD files in the proper locations and checked that their permissions are legal, it's time to see if all of this hard work will be rewarded. Reboot!

Topic 57. What does it look like when everything works properly?

```
-------------------------------------------------------------------------------

________________________________________________________________________


>b le()
Using IP Address 192.168.0.150 = C0A80096
Boot: le(0,0,0)
Booting from tftp server at 192.168.0.1 = C0A80002
Downloaded 43784 bytes from tftp server

>> NetBSD/sparc Secondary Boot, Revision 1.7
>> (pk@flambard, Fri Jan  2 00:10:58 MET 1996)
Using IP address: 192.168.0.150
Boot: client IP address: 192.168.0.150
Boot: client name: sphere
Boot addr=192.168.0.150 path=/usr/export/sphere
Booting netbsd @ 0x4000
1376256116800+[77844+91083]
.
.
.
Enter pathname of shell or RETURN for sh:
________________________________________________________________________r
```

Success! sphere is:

1) Loading the MONITOR program from EEPROM
2) Running self-checks to guarantee machine "ready" state
3) Executing the Primary Stage Boot Loader (PSBL) Program
4) Asking for its IP address via a RARP request
5) Obtaining its IP address from the Linux RARP server
6) Downloading its SSBL from the Linux TFTP server (43784 bytes)
7) Obtaining its name from the Linux bootparamd (sphere)
8) Obtaining kernel and file system pointers from bootparamd
9) Using NFS to download the NetBSD kernel
10) Using NFS to mount its file system
11) Executing the NetBSD kernel

The Sun has booted! Mission accomplished.

I hope that this document has both entertained and helped you.

It was a pleasure to write.

g.

Bibliography

Innumerable documents provided the resources necessary to enable me to create this text. I am very grateful for their existence, and that is why I wrote this document

----

Revision History

CREATED: 1997-03-01

```
UPDATED:       2017-03-11

-------------------------------------------------------------------------------
END:  Netbooting a Sun 4/380 on NetBSD via Linux HOWTO
-------------------------------------------------------------------------------
```