Global Information Assurance Certification Paper

Network Security In A Thin Client (LTSP) Architecture

Jonathon M. Robison

Submitted for completion of the GIAC GSEC
Practical Version 1.4b
April, 2004

Table of Contents

ABSTRACT

Many organizations, both large and small, are investigating the potential of thin client architectures for their companies. In general, a thin client is one which does not have any local storage. There are ranges of thin clients, such as what one could call ultra-thin (no CPU, no fans, no storage, no memory), all the way up to systems that are essentially a normal desktop system of today with the hard drives removed (such as the Dell SX260 and similar lines).

The guiding wish behind such investigations is that by moving the majority (if not all) of the operating system, as well as the data storage, away from the desk to a central location, system administration costs can be drastically reduced. Although the majority of the Total Cost of Ownership (TCO) information for thin clients is anecdotal, it isn't a tremendous reach to assume that any time you don't have to physically touch the client hardware, you save money.

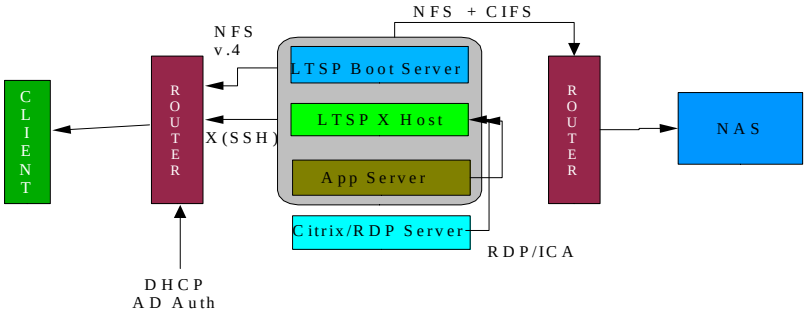
One of the most visible up and coming solutions for a thin client architecture today is the Linux Terminal Server Project (LTSP)[i]. Although it is designed for serving thin clients with a Linux desktop, it can also be used to serve a Microsoft Windows® desktop, using the LTSP as nothing more than a boot and transport mechanism. The Open Source developers of LTSP have even gone to the trouble to include scripts to automatically bring up a Windows desktop via rdesktop (or a Citrix[ii] ICA Client) when a user boots.

The typical LTSP solution involves a number of protocols, particularly DHCP, TFTP, NFS, and lastly, X. When a Microsoft Windows desktop is desired, additional communications such as RDP and ICA (as examples) are added.

This practical assignment will focus on the non-Microsoft portions of the solution. Principally, DHCP, TFTP, NFS, and, perhaps most importantly for this case, X.

DIAGRAM

The following is a simplified diagram of a typical thin client solution. It is by no means the only architecture which can be created, but is typical enough to use as a reference for a discussion of network transmission security.



PROTOCOLS

DHCP

Dynamic Host Control Protocol, or DHCP, is one of the most common means in use today for dynamically assigning IP addresses to clients as they connect to the network. In many cases, the server will assign an IP address to any client which asks for one. In a slightly more secure setup, the DHCP server can be set up to deny an IP address to any client whose MAC address is not known, and if

desired give a specific IP address (and often an associated hostname) to those known MAC addresses. While this provides some measure of protection, another “layer” to defense, (Defense in Depth), it can do little to prevent clients who can spoof a MAC address from connecting. In a thin client arena, this is less of a worry, as the MAC spoofing generally is done *after* an OS is up an running, not during the early stages of the OS boot process. Since thin clients get their OS from a central server, the user has little opportunity to alter their MAC address. Of course, this does little to prevent a user from walking into the office with his personal laptop and connecting to the network, if he or she chooses to spoof a MAC address.

Possible additional tools in the security arsenal in relation to DHCP would be the implementation of Secure DHCP with DNSSEC[iii], although there are no public implementations of this searchable on Google.

Because most client requests are 'broadcast” requests (sent to address 0.0.0.0, and thus “heard” by all network connections on the network) it is a trivial matter for a non-trusted DHCP server to answer the request and provide network information to the booting client, as well as a non-trusted kernel image for the client to boot from (using tftp). Because of this broadcast nature, most enterprises choose to have their routers block broadcast requests outside of the subnet from whence they originated. Generally, therefore, an attacker who wants to emulate a DHCP server will need access to the subnet itself.

TFTP

Tied to DHCP very closely is TFTP, or Trivial File Transfer Protocol. In the thin client arena, this is often what is used to transfer the Linux kernel image to the thin client device so that it can boot beyond the BIOS environment. It is what it says it is, trivial. In most cases, the TFTP protocol is attached to the DHCP server, and it is the DHCP server which serves up the kernel image. The security of the TFTP session is therefore entirely dependent on the security and trustworthiness of the DHCP server.

BYPASSING DHCP AND TFTP

One possible method of getting around the need for extensive DHCP security is to use a boot device, such as a mini CDROM or a USB key fob (if the hardware BIOS will support booting from a USB device). In many enterprises, the blocking of broadcast requests at the router level can prevent a network boot request from working. Having a key fob (USB) boot device with the kernel image already on it prevents the need for modification of the infrastructure in those cases. It also reduces the ability of an attacker to masquerade as the DHCP server and thus send untrusted kernel images for the thin clients to boot.

The key fob, or mini-CD, contains the entire kernel image, as well as IP address information, netmask, etc. This means that a user's IP address moves with them, from thin client machine to thin client machine, rather than being dynamically allocated. This does mean an increased need for physical security, however, in exchange for the reduced security need in the area of DHCP/TFTP servers. Physical security is discussed later in this practical.

In some enterprises, IP addresses are tied to physical areas within the company. In those cases, the IP address cannot “go with” the user. In addition, in many cases the department or group which manages the DHCP server may refuse to add TFTP to them, thus making them useless as a thin client boot system. For those cases, the syslinux.cfg file of the key fob can be set to use normal DHCP requests to obtain network information, and have the IP address of it's LTSP server hard coded. Examples are available within the mini-CD iso image at http://www.ltsp.org's download area.

Wherever possible, I would recommend the use of a key fob or mini-CD for booting thin clients. This places the least amount of exposure on the network.

NFS

[Large portions of this NFS section are taken in whole or in part from the excellent work by Nawapong Nakjang Banchong Harangsri listed in Endnote iv]

NFS, or Network File System, has a long history in Unix and Linux. “*The main problems with NFS are that it relies on the inherently insecure UDP protocol, transactions are not encrypted and hosts and users cannot be easily authenticated*”.[iv]

The following are some actions which can be taken to increase the security of an NFS client/server setup.

1. The NFS file system on the server should be on a separate partition or disk. This prevents malicious users from filling up the entire OS file system and locking up the server.
2. Prevent normal client users from mounting an NFS share by using the “secure” option in the server's /etc/exports file.
3. Export NFS file systems with the appropriate permissions (i.e. use (ro) rather than blank or (rw) in /etc/exports.
4. Restrict exporting of NFS file systems to a specific set of client applications on the server.
5. Specify only a specific set of NFS clients that will be allowed to mount an NFS file system. If possible, use numeric IP addresses or fully qualified domain names, instead of aliases.
6. Use the 'root_squash' option in /etc/exports on the NFS server if possible

7. Disable suid (superuser ID) on an NFS file system (on the client). This actually protects the client from malicious code on the server more than the reverse.
8. Install the most recent patches for NFS and portmapper (on client & server)
9. Encrypt NFS traffic using SSH (on client & server) [Although it is too long and perhaps too detailed to include here, I highly recommend reviewing Mr. Harangsri's instructions on encrypting nfs with ssh which can be found at http://www.linuxsecurity.com/feature_stories/feature_story-118.html.]

X

The majority of the traffic in an LTSP-based system is X traffic. The server is normally accessed via XDMCP, and XDMCP provides a login session. Once all that is begun, everything the user sees and does is transmitted over X protocol between the client and server. Irregardless of the security of the NFS connection, the traffic passing over X is the one that an attacker will want to sniff. Therefore it should be the system architect's prime target for security of the thin client system he designs.

A short list of known problems with X[v]:

1. Setuid clients and scripting toolkits
2. Connection setup and sniffing (primary concern for thin clients)
3. Server is often setuid root

X itself, running on any system, has a large number of security issues. We are not targeting those issues here [see Endnote V for a good review of X issues], but rather concentrating on securing the transfer of data from one system to another.

Access control is done mainly at the transport level. Once a client has a connection, he can have full control over the X server. Authentication types (xhost, Xauth, XDM, etc.) are relatively weak. The true problem with any of these is that once a connection is established, the remaining data transfer is essentially clear text. Data from any thin client is then subject to monitoring, man-in-the-middle attacks, and any other form of data watching imaginable. If a systems architect was to propose to the CEO of a company that he perform all his desktop work (email, contract review, salary discussion, etc.) on a huge 50' x 40' monitor located in a public parking lot, it would be equivalent to what is done when thin clients are used without encryption of the data in transport.

One workaround for this is to tunnel X through SSH. Whereas firewalls may complicate this issue, in general it is not expected that corporate thin clients will be booting up through the firewall. If you need to boot thin clients through a firewall, then you should look into hardware or software VPN systems (which are beyond the scope of this practical).

SSH also has the practical benefit of be able to be compressed, reducing bandwidth requirements on 10M and slower networks. (100M and 1G networks need have less concern with this issue, but they are certainly free to compress their X data if they wish).

However, if you try and use SSH keys, problems arise. As a test, try booting your Linux system into init 3 (character mode), and typing the following:

xinit /usr/bin/ssh -C OTHERHOST gnome-session

If, and only if, you have already set up the ssh keys on the client and server, placed the client's public ssh key into the authorized_keys file(s) of the server, added your clients public key to the server's known_hosts file, etc. will you have success. If you are like the rest of us, you will have to flail and pull some hair out before you will finally see the login screen from the server.

Thus, incorporating SSH keys into the LTSP process is not something to be attempted lightly. Work is currently under way in the open source community to incorporate X over SSH into the default operation of LTSP, alleviating some of the more tedious aspects of the process. Initial testing has led to promising results for using SSH without sharing keys.

In a scenario where X is tunneled through SSH using keys, it is important to understand that the security of the server is of paramount importance. Doubly so, since the client ssh software (and the keys) are also on the server. Remember that in LTSP solutions, once network communications have been established by the boot kernel, a pivotroot is performed and the directory /opt/ltsp/i386 (usually) on the server is now the root (/) directory of the clients, via NFS. Thinking this through, you should realize that both the public and private keys, as well as the client's ssh binaries are all actually located on the server. If that server is compromised, the entire system is breached.

I mentioned earlier that I preferred using USB key fobs and mini-CDs for a boot device, and this is one of the reasons why. It is possible to have the key fob mounted on the client immediately after the pivotroot (or before, if you feel brave enough to try NFS v4 or NFS over SSH), thus providing a storage location for ssh keys which is independent of the server. It is even possible to have different keys for each user (which should be the preferred method) but understand that this will involve more effort when setting up the server's authorized_keys and known_hosts files. In addition, having keys on the key fob or CD becomes a huge security issue as well.

GETTING AROUND SSH KEYS

If you weren't confused enough, there is another approach to using SSH for the X data which does not involve pre-existing keys.

Backing up a moment, consider the following: LTSP is, in essence, a mini-distribution of Linux which is provided (normally) via PXE boot or Etherboot, or in some cases a CD or key fob. All this mini-distribution needs to do is get the kernel running, an NFS client, and the client's local X server. We have continued to think of this thin client as nothing but a glass screen for the server account we want to log into. However, recognition of what it really is opens up another possibility. We could start X on the client, and instead of jumping directly into remote display scripts, we open an xterm for the sole purpose of performing the ssh command. Instead of passing the ssh command to the xinit process on X startup, we instead perform that command in the xterm. Doing this allows us to answer the password question given by the server when the ssh keys are not pre-existing and setup for prompt-less login. It also allows the process to use whatever login systems (pam, using any number of pam modules) exist on the server. Credit for this “big picture” approach goes to James Geddes (co founder of x-windows) of HP.

An industrious programmer could even create a pretty graphical client instead of an xterm to request the username and password to use.

These are just a few possible solution to using X over SSH. There are other efforts under way to make X more secure, and other methods of keeping keys. Some of these can be reviewed at http://www.xfree86.org. Each architect must evaluate the suitability of various technical solutions for themselves. Architecturally, any solution which provides at least a moderate level of encryption for X traffic is better than none.

KERBEROS IS YOUR FRIEND

Lastly, another option exists which could perhaps ease troubles across the board. I have to take credit for this one. Consider the option of stopping the LTSP boot process immediately after the network interface is up and running, prior to mounting the new root file system from the server's exported directory. What would happen if the user was prompted for a username/pass combination at that point, and authenticated against a directory server which provided forwardable kerberos tickets? The ssh tunnel for NFS could be set up using the ticket, and the X over ssh could use the same ticket[vi].

As soon as this method becomes available, it should be used. The simplicity of design, the usefulness of the kerberos ticket, etc. all point to this as the most 'elegant' solution. Until then, I recommend using the “login at the client” method listed above in the “Getting around SSH keys” section.

DO I STILL NEED XDMCP IF I USE X OVER SSH?

The answer is no. The use of SSH to initiate an X session removes the need to

enable the XDMCP server on the server. While not a heavily attacked subsystem, running any service which isn't required is an invitation to crackers. Therefore, the ability to shut off XDMCP on the server is a plus.

SEPARATION OF CHURCH AND STATE

As mentioned in the protocol section, above, in most thin client architectures (especially those based upon LTSP), the security of the server is paramount. If that server is compromised, the cracker has full access to the system. Using keyless SSH and/or kerberos can minimize that risk, as the keys or user information (including password) is not stored on the server and is instead stored elsewhere.

Further safety is gained if the user's home directory is mounted at boot time (through the use of something like pam_mount[vii]) from a file server seperate from the LTSP server so that user directories are not available to a cracker even if he succeeds in gaining access to the LTSP and/or application servers. For the truly paranoid, pam_mount can even mount encrypted home directories.

Physical security becomes even more important for enterprises who choose to attempt the ssh key sharing through a medium such as a key fob (not the recommended method of using SSH in LTSP). The physical security headache just expanded exponentially from one (the server) to every single user. This is yet another reason not to use the shared key approach to ssh for thin clients. Physical security of servers is an entire article in and of itself and is considered out of scope for this document.

NETWORK QUARANTINE

In many enterprises, the need often arises to provide network connectivity and yet still prevent access to the corporate intranet. For example, when a vendor comes into the corporate offices to perform a demo, they may need to access the external Internet in order to perform the demo. In many cases, this cannot happen because in order for them to access the Internet, they must access the intranet first.

Network Quarantine is a simple concept. If you are unknown, you get shunted off to a separate area for untrusted systems. The most common example of this is in hotels around the USA, many of which use STSN[viii] as their service provider to the hotel rooms. Normal DHCP works, and provides an IP address. However, any web address you enter gets redirected to their payment page. Until you pay, your IP is kept part of a VLAN in which all requests are either ignored (for non-HTML traffic) or redirected to their web page. Once you pay, your IP is moved off the “locked” VLAN and into a VLAN with a proper gateway to the rest of the

world.

Because the thin client architecture should be consistent across all end user client devices (since they all boot from the same image) it is relatively simplistic to do checksum comparison during the login process. This checksum can be done at any time during the boot process [Note that if clients use different mount points for pivotroot, the checksum should be done prior to the pivotroot], perhaps incorporating an MD5 hash of the checksum, the date and a hardware spec (such as the MAC address) which can be quickly performed by both the client and server, permitting at least a modicum of trust that the system connecting to the network is truly a corporate OS load. While this is not perfect (a cracker could drive a truck through the security holes in this arrangement) and it falls apart in any environment where the signal passes through a switch and the switches MAC is what is shown[ix]) it is not meant to stand alone, but rather to provide one layer of Defense in Depth.

Major hardware vendors such as Cisco are starting to work on hardware to support and enable network quarantine. These hardware solutions are not yet productized.

Although included in this practical, network quarantine is not yet mature enough to include as a final recommendation. Companies that are looking to use thin clients, however, should also begin reviewing the technology and state of network quarantine and perhaps implement them both together.

ALTERNATIVES TO LTSP

Although the focus of this Practical is to examine LTSP-based solutions utilizing open source wherever possible, a large majority of the network security issues have already been addressed by proprietary vendors.

SUN MICROSYSTEMS SUNRAY

Sun Microsystems has a product called “SunRay”[x] which is essentially a dumb terminal, having little more than some firmware in BIOS and a graphics card.

SunRay uses the SLIM protocol, which incorporates encryption into it, as well as compression. The keys for this encryption, if any, are in the firmware. Because this firmware is not accessible, and the keys are never transmitted over the wire, the security of the SunRay systems communication is excellent. The SunRay also provides user authentication and session management. The addition of 'Smart Cards' to the SunRay systems can make them an enviable choice for many enterprises, as a user can pull his Smart Card while applications are still open and running, insert it in another SunRay elsewhere, and instantly his session, including open applications, is presented on the new device. LTSP will

not reach this level of corporate readiness for some years yet.

In addition to solving the security troubles, Sun has already solved other issues, such as access to locally connected devices, such as a USB Palm Pilot, etc. Currently, LTSP is only just now experimenting (with help from major vendors) with making such things work on a thin client.

At this time, the SunRay system provides only a Solaris desktop, not Linux. However, the SunRay server software is currently being ported to Linux and should be available in late 2004. Until that time, corporations can use Solaris to serve the SunRay's clients and the SunRay login process can be re-directed to a Linux system. Security for this arrangement is questionable, however and is suitable only for testing until the true SunRay server on Linux is available.

NEOWARE, HP, WYSE

Various vendors design and sell thin client devices. In all cases, they are primarily Windows CE or Windows XPE devices. All claim that the devices can boot to a USB key fob (they have no CDROM drives, generally) and using PXE, thus making them LTSP compatible. However, when booted in this manner, their differences from booting a standard PC with LTSP are negligible, if at all, and thus will not be covered in this practical.

CONCLUSION

In summary, perhaps the greatest friend to an enterprise LTSP implementation is SSH. If a Kerberos Domain Controller (KDC) is available, then obtaining a Kerberos ticket early in the boot and using it for an SSH tunnel for NFS, as well as the SSH call to execute the X session on the server is recommended. If kerberos is not available, then a more normal SSH tunnel setup should be attempted for NFS, and an xterm on the client during boot to initiate the X over SSH.

PXE/Etherboot should be used to obtain the boot kernel if the company routers have IP Helper installed, permitting broadcast request through the routers and allowing TFTP to work, and presuming the IT management does not want to added security (but additional headaches) of distributing CDs or USB keyfobs. If not (many companies do not allow this) then USB keyfobs or Mini-CDs is the way to go.

---

[i] Linux Terminal Server project, http://www.ltsp.org

[ii] Citrix – Access the On-Demand Enterprise, http://www.citrix.com

[iii] Lemon, Ted & Richardson, Michael. “Securing DHCP DNSSEC bourne public keys”. February 23, 2003
URL: http://www.sandelman.ottawa.on.ca/SSW/ietf/dhc/draft-richardson-dhc-auth-sig0.html

ivNawapong Nakjang Banchong Harangsri, “NFS Security”, September 6, 2002
URL: http://www.linuxsecurity.com/feature_stories/feature_story-118.html
vHerrb, Mattheiu, “Enhancing Xfree86 Security”, July 2003
URL: http://www.openbsd.org/papers/xf86-sec.pdf
viDaniel J. Barrett and Richard E. Silverman, “SSH, The Secure Shell; The Definitive Guide”, Sebastopol, CA, O'Reilly & Associates, 2001 p412
viiPAM_MOUNT, A PAM module that can mount volumes for a user session. Mike Petullo
URL: http://www.flyn.org/projects/pam_mount
viiiSTSN, Secure Broadband to go for Business
URL: http://www.stsn.com
ixOUR TEXTBOOK – EDIT THIS!
x“White Paper – SunRay Overview”, April 2003
URL: http://www.sun.com/sunray/whitepapers/SunRay_WP042403.pdf