Global Information Assurance Certification Paper

Author: William D. Pool
Date: 02-26-01

It has worked all these years why change it?

With the amount of security threats and holes being encountered in the IS field within the last decade, we still hear, "It has worked all these years why change it?". With companies that grew in an era that was not too concerned with security, some of their administrators and network engineers grew with the same mindset. Those that choose to become more security conscious will most likely see harm coming their way and be more aware to avoid it. If they haven't gained an interest in security they might be forced to realize that the old way does not work anymore. Some examples of security practices can involve hardening a machine by stripping the operating system to a bare bones minimum for a server (i.e. removing unneeded operating system packages and/or software), locking a service that posses a huge security threat and services running that are not needed on your network, and keeping up with security patches, vulnerabilities, and advisories. What if the threat were so great that it could compromise the (UNIX) core of the IS department of the company, leaving the company at the whim of a cracker and, even worse, crippled with scrubbed or corrupted servers? At that point, what else could considered? The company would be panicking because they would be unaware of the cause of the problem at hand. While in a rush to fix it, the thought that "It will never happen to me" would be fading, if not already completely out of mind. At the same time, the cracker would be attacking the company across the street from you from your compromised machine. You would eventually clean everything up and think you had the cracker out. You would restore backups of the systems, unsure if those backups had compromised data on them (depending how long the cracker had been in your systems), but to you, everything would be safe; you would be in the clear; you would be back in business. A few hours later, you would get a call from a company saying your company had attempted to hack their network. They would be talking about lawsuits and legal battles. What would have just happened? What could have caused this nightmare? All of this a hellish dream or reality (depending on how you may look at it) from an older way of thinking. If you have had taken some basic security measures, added strong authentication, encrypted communications channels, better host verification, and have knowledge of current security vulnerabilities, topics and issues, this may never happen.

Most people don't think the above scenario would ever happen to their company. They believe they are too small to have a cracker attack them the thought of security by obscurity. Not many people think there could be an intruder watching them without their knowledge on their internal or external network. According to many online experts (Securityfocus.com, SANS.org, securityportal.com, cert.org) 80% of computer security incidents occur inside the company. In the past, people have used various vulnerabilities and exploits to gain access to systems.

A popular example in the media today (probably over glamorized) is Kevin

Mitnick. Kevin Mitnick was a hacker that was wanted by the law on several accounts of penetrating companies and obtaining proprietary data. Other accounts were phone tapping and impersonating company officials. He penetrated Tsutomu Shimomura's personal computer. Tsutomu Shimomura was helping the FBI capture Mitnick and, being well known as a security expert at the time, provided Mitnick with a challenge. Mitnick gained access to the machine by using host spoofing (TCP/IP exploit) and then gaining a trust relationship of Tsutomu Shimomura's machines using the r-commands (rlogin, trust exploit).

We can get even more basic in explaining the source of the problem that allowed the hack listed above; poor authentication, and clear-text distribution of data of the services. These two things are probably the most underestimated security breech into a system today. The primary services that present problems are:

- Inetd - Internet Services Daemon
- Telnet - Service Port 23
- FTP - Service Port 21
- RSH (rsh, rlogin, rcp) - Service Port 514
- sunrpc- Service Port 111

The main problem with *Inetd* is that it does not offer reverse mapping of IP addresses to the hostname to validate it. In essence, you can allow a false IP or hostname that is NOT registered with a proper DNS entry to connect and function. This can allow spoofed connection to the desired service. When looking over the functions allowed on *inetd* administrators could disable services that are not needed. This will help close known holes. On most default stock installations of UNIX, there are many services open that are **not** needed. Getting rid of the services can be accomplished by commenting the services in **/etc/inetd.conf**, then restarting *inetd* for the new settings to take effect.

For someone new to commenting out services, let us take an example using a stock **/etc/inetd.conf**. Below shows a stock entry for telnet in **/etc/inetd.conf** then a commented entry for that service to disable it.

- telnet stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd
- #telnet stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd

One software program, *TCP_Wrappers*, can be used with the traditional *inetd* to make the default services more secure. This is an excerpt by the author (Mr. Wietse) of *TCP_Wrappers* software package describes its function:

> *"These programs log the client host name of incoming telnet, ftp, rsh, rlogin, finger etc. requests. Security options are: access control per host, domain and or service; detection of host name spoofing or host address spoofing; booby traps to implement an early-warning system."*

As stated above, *TCP_Wrappers* can cause a *low* end host verification and

authentication to the service. When *TCP_Wrappers* is installed, to enable it in **/etc/inetd.conf**, change the line of the running service; in this example, we will assume *TCP_Wrappers* is installed in /opt/GNU/sbin and the service we will change will be telnet.

telnet stream tcp6 nowait root **/opt/GNU/sbin/tcpd** in.telnetd

In conjunction with *TCP_Wrappers*, another piece of software could be installed to replace the traditional *inetd* daemon; this *new* software is *xinetd*. Some of the features of *xinetd* include: access control, prevent denial of service attacks, extensive logging abilities, offload services to a remote host, IPv6 support and user interaction.

Another widely used service, *telnet* is used to remotely communicate between most UNIX machines and is still very popular. It is a direct host to host connection via port 23 that has no built in security measures put in to place to prevent an intruder. The biggest problem with this service is the fact it is not encrypted. This causes a risk if you were to have someone sniffing your network. This would allow them to view user ID, password entries into the system, important documents being transferred, and even conversations over the internet. It would also allow them to gain knowledge of what you are doing on the system.

FTP is clearly one of the most widely used service for file transfers over the internet that its share of problems with security as well. The client issuing commands to the server that is listening on the default port of 21. The actual file transfer has the client instructing the server, IP and dynamic port number to use. At that point the client instructs the server to transfer the file. The server opens a TCP connection to the client's address and port with the source (server end) port number of 20. While the IPs are communicating, someone could spoof the source address of the client *accessing* the FTP server allowing the cracker to assume that client's IP. To clearly see the workings of FTP, the following example is given:

(client establishes connection from local port 1024 to server port 21)
(client listens on port 1025 (4,1))

client:1024 -> server:21	*PORT c,li,e,nt,4,1*
client:1024 <- server:21	*200 PORT command successful*
client:1024 -> server:21	*RETR file*
client:1024 <- server:21	*150 Opening ASCII mode ...*
client:1025 <- server:20	*<data for 'file'>*
client:1024 <- server:21	*226 Transfer complete*

The R-commands (RSH, RCP, RLOGIN), work in the following matter:

(Client attempts to assign a local port <1024, eg 1023)

client:1023 --> server:514	*<connects>*
client:1023 --> server:514	*<nul>luser<nul>ruser<nul>cmd<nul>*

client:1023 <-- server:514 <nul>
client:1023 <-> server:514 <data>

These communication methods are not secure; communication is being transferred as clear-text over the network between the machines. A lot of programs today *still* use clear-text for a lot of their communication. One example is *Legato Networker*. This backup program uses r-commands to send data between clients and control the hosts who have the backup software. *If* someone were to develop a way to exploit the communication barrier, they could possibly have control of the backup server and all of its data. This would cause a security hole in the *Legato Networker* program.

The RSH authentication scheme uses non-safely implemented host to host trust. A user would use an "rhost" file to say that "root@some-box" can have root access on "server." The "server" would then look at the hostname given by the connecting box to authenticate it. The problem is that this method makes the critical, and wrong, assumption that an intruder can not spoof the host on the source machine or in transit to the other host. This exists today and could be easily done if the system were setup poorly. This could very well cause a root compromise or a privileged user account could be exploited. This could be a huge risk especially if root were to be allowed in the "rhost" files on all the clients and or server machines.

With the services mentioned, using clear-text would allow someone to watch or view the traffic on the box with a sniffer. Take note that even if you use encrypted communications, if your machines are root compromised, they can still sniff *all* connections on your network. This would give the cracker an upper hand in your defense against them. *Some* tools that could be used for such an attempt would be *NetCat*, and *dsniff*. With the increasing amount of people in the world who use *ftp*, *telnet* and the *R-commands*, none of the data being transferred is being encrypted, unless by the software listed below.

Most UNIX flavors still leave services open; legacy services which have potential for some major damage. So, how does someone plug these holes up? How does someone still keep the compatibility with how the company been doing things *forever*? One solution in the internet community is called "*Secure Shell*". *Secure Shell* is a drop-in replacement for the clear text services, mentioned in this paper, on the internet in the UNIX world. Those services are *telnet*, *ftp*, and the *r-commands*. It provides an encrypted method of transport and authentication to make sure the right person is accessing the box. *Secure Shell* can also be used in conjunction with *TCP_Wrappers* for better security. The supported methods of encryption for *Secure Shell* (*SSH*) are DES, 3DES and blowfish. You can remove the *r-commands* completely, or make them backwards compatible with *SSH* as a drop in, allowing *SSH* to be used when possible; if not, *RSH* can be used. Until migration has been moved over to the full *SSH* solution, you can keep the traditional method of host based maps for authentication so scripts dependent on *RSH* commands are compatible. When using *SSH*, you can use encrypted host based maps, that work the same way as the traditional *RSH* maps, yet secure. We've mentioned earlier in this paper the use of *TCP_Wrappers* for host based verification to do

DNS verification.

How does one go about using *Secure Shell* into a native *RSH* environment? How does one go about using *Secure Shell* to replace *ftp* and *telnet*? *Secure Shell* can be installed with or without *RSH* compatibility. If you are not using *RSH* commands, then your best bet would be to not even worry about allowing compatibility for them. In essence *Secure Shell* would limit the ports being opened and the amount of different programs being used. The primary program in *Secure Shell* is the *ssh* program. This works in a similar syntax to *rlogin*.

ssh -l <username> <hostname>

Using one standardized port for *Secure Shell* (port 22) would tighten security from an outside view of the box. Instead of having at least four ports open, you would have only two (*sftp*/*sshd*). You can set the encryption key on *Secure Shell* to 768 or any higher value appropriate for your needs (and legal requirements of the region). *Secure shell* also includes a program call *scp* which is a secured version of the traditional *rcp* command. *Secure Shell* also provides a *sftp-server* which is an encrypted version of the *ftp* daemon. Another secure *ftp* software program available is *safeTP*.

Another security precaution companies might want to look into would be hardening the machine. This is a part of the above mentioned security measures. What about all the unnecessary installed operating system packages/software on the machine? A majority of companies install the full *stock* version of the operating system. Even when an administrator installs just the *core* Operating System on a machine, it can probably be stripped down to match the machines actual needs. This varies from vender to vendor on the Operating System in question. Not many companies believe in patch or system upgrades. One thing you see all over the inside of companies is the use of X-Windows on both their desktops and on their critical servers . Most the time the Administrators only go near the console occasionally. Leaving X-Windows on a machine leaves *rpc* service open to vulnerabilities, as well as causing an extra resource for the server that it does not need. There are still a great number of insecure machines on the net that have not yet upgraded to the latest Operating System released or the latest patch released and take basic security measures. Most of these places, ironically, are, but are not limited to, government and educational systems on the internet. With the increasing amount of users connecting stock UNIX machines on their ISDN and DSL lines without thinking of security, leaving more vulnerable points for a cracker. It only takes an internet connection and a knowledgeable cracker to take down even the most advanced companies who might be thought of as secure.

After being given an insight of *clear-text* services, that are insecure in nature, there are some secure alternatives like *Secure Shell* and *safeTP*. Hardening a machine is crucial in the effectiveness of your systems performance and intended use. This will limit the risk factor of unwanted and unneeded security holes and vulnerabilities within them. In today's society, one needs to have encryption, good authentication, and an awareness of security. Taking care of insecure services early in the game and hardening

the operating system will help reduce problems in the future. When implementing the things mentioned: *xinetd, tcp_wrappers, Secure Shell, and hardening of the box,* provides a better means of communication between machines, as well as making it secure. Remember security ***DOES*** affect your company and its productivity. Security affects confidentiality, integrity and availability. Without someone of a conscious mind of security watching your systems and your back, who is there to protect you when the kingdom falls? Here is a final analogy;

> "Humpty dumpty sat on a wall, humpty dumpty had a great fall, all the kings horses and all the kings men couldn't put humpty together again."

References

1.??Security Vulnerabilities between FTP & Berkeley Rsh/Rlogin Protocols

URL: http://www.daedalus.co.nz/~don/ftp.html (02-22-01)

URL: http://securityportal.com/cover/coverstory2000814.html (02-22-01)
3.)Wietse's Collection of Tools and Papers
URL: ftp://ftp.porcupine.org/pub/security/index.html (02-22-01)
4.)OpenSSH
URL: http://www.openssh.com (02-22-01)
5.)Xinetd
URL: http://www.xinetd.org (02-22-01)
6.)safeTP
URL: http://www.cs.berkeley.edu/~smcpeak/SafeTP (02-22-01)
7.)Ryan Russell and Stace Cunningham. Hack Proofing your Network Internet Tradecraft. Syngress Media, January 15, 2000. ISBN 1928994156. p146, 260-264, 279, 285-296 - ISBN: 1928994156