UnixWorld Online: Internet Security: Column No. 002

Trusted Hosts

By Rik Farrow

Becca Thomas suggested that I explain what I meant by trusted host in my last column, which turns out to be as much of a hot topic as ever, and a prime way to break into many UNIX systems.

Questions regarding this article should be directed to the author at *rik@spirit.com*.

The trusted host problems had their genesis way back in the early '80's, when the Computer Science Research Group at University of California, Berkeley Campus was commissioned by DARPA (Advanced Research Projects Agency, part of the Department of Defense), to improve the UNIX system. A large part of this project was porting the TCP/IP protocols and applications to the UNIX system and the C language, a very important event in itself.

Before the Telnet application was completely ported, some simpler applications were written to permit remote logins and execution of programs. These programs--known collectively known as the ''r'' commands--consist of `rlogin, rsh, rcp`, and some others. It is the first three which concern us here, because of the trust problem.

In a university environment, especially in the early eighties, when computers were still rare and there were only 200 sites on the ARPANET (pre-Internet), security was not a big concern. But convenience was so the ''r'' commands are big on convenience and short on security.

Let's imagine that there are three systems named violet, daisy, and boffo, all on the same network, and that the administrator of violet trusts daisy, and vice versa. The configuration file that spells out this trust arrangement is named `/etc/hosts.equiv` (see Listing 1). Now, any user with an account on both systems can login into his or her account on the other system without re-entering a password. The assumption is that if a user is logged on to a trusted system, he or she should not be required to authenticate by using a password a second time.

The `rlogin` command passes along the names of the remote user and the remote host, and the local `rlogin` server (`rlogind`) uses this information to determine whether the user is pre-authenticated or if a password should be required. Commands other than `rlogin` will not even prompt for a password if trust has not been extended. That is, if you want to use `rsh` or `rcp`, you must have setup trust between hosts.

In some cases, a user might have a different name on each host. The ''r'' commands provide for this too. Each user can have a file named `.rhosts` in their home directory. The user places host name-user name pairs in this file. For example, a user named rik on violet but rfarrow on daisy might include a line like ''daisy rfarrow'' so that he could log into his own account on violet without entering a password. In the case of the root (superuser), the `/etc/hosts.equiv` file is always ignored, and only `/.rhosts` is checked.

Danger!

Hopefully, you are already beginning to notice some red flags flying. For one thing, allowing users to create their own `~/.rhosts` file means that any user can share his or her account with any other user with no password. Sharing accounts is never a good idea, and letting your users arrange this on their own is an even worse idea. I suggest that you decide whether to allow this or not, and make it part of your local security policy. You can enforce this by having the `cron` daemon run a shell script which looks for `.rhosts` files in all home directories, mails a copy of any such file to root, remove the file, and mail a warning to the user.

Even arranging for trust with the `/etc/hosts.equiv` file is not safe. Remember that I said there were three machines, but have so far left out boffo. Suppose the administrator of violet doesn't trust boffo, but daisy's admin does. So a user from boffo can log into daisy, and then from there log into a similarly-named account on violet-- even though violet's configuration file would deny this. Trust is transitive. When setting up a trust arrangement, it is essential that trust not be extended to *any* system outside the local ring of trusted systems.

Sun Microsystems has participated in one of the darker moments in UNIX system security. In all releases of SunOS, including the latest (4.1.4), and the first several releases of Solaris 2.x (but not including 2.3), the `/etc/hosts.equiv` file included with the distribution contains only a single visible character, the plus sign (+). The plus sign is the wild card, and means that a newly installed Sun system running SunOS trusts *any* user (except root) on any host. While most systems won't have an account for rik or guest on them, they will have standard accounts, such as bin. Thus, an intruder could log into any Sun system (which hadn't been correctly configured) as the bin user, across the network, with no password.

Was Sun trying to make their systems insecure intentionally? No, although they certainly succeeded. Having the plus sign just made newly installed Sun workstations easier to use on a network. If you have Sun systems, check for the plus sign in `/etc/hosts.equiv` anywhere in the file by itself at the beginning of a line. A plus sign in front of a hostname is okay (but not necessary). The plus sign works the same way on any system that supports the ''r'' commands, and you should delete it if found (or trust everyone). A minus sign denies access. You could have a `/etc/hosts.equiv` file with just a minus sign, and deny access.

Of course, if this were the only problem, things wouldn't be too bad. The `rlogin` command does have a bright side. When trust is extended, no password is sent across the network, so password sniffing programs will fail. You might also be using ''r'' commands locally to aid in system administration. What you should not do is allow use of the ''r'' commands between networks which you do not control and therefore, can't trust.

Internet Invaders

Someone who wishes to break into a UNIX system will often search for trust relationships. Kevin Mitnick did this (see my last column), and this is not uncommon. Or suppose that a person running a password sniffer captures the packets showing the use of `rlogin` without a password. The sniff file includes the name of the local and remote hosts, and the account name. To login, all they have to do is fake his or her own host name and account name. The ‘‘r’’ servers rely on the names found in the `rlogin` request to determine if the request will be permitted or denied. If the server blindly trusts this information, then the attacker can login with no password.

Modern ‘‘r’’ servers include a reverse name lookup. That is, they use the source IP address and convert it to a hostname using `gethostbyaddr()`. If the hostname discovered by converting the IP address doesn’t match the name in the `rlogin` packet, access is denied. Sun systems do this, but is it sufficient? Nope. The attacker probably controls the remote system, and can tell the Domain Name System there to lie (see Steven Bellovin’s paper ‘‘Using DNS for System Break-ins’’ in the *Proceedings of the Fifth Usenix Security Symposium*).

Or, in another attack, the attacker uses initial sequence number guessing (see my last column) and sends packets to the ‘‘r’’ server without ever receiving a reply. This is the form of IP address spoofing used by Kevin Mitnick in his Christmas Day attack.

There are a couple of things you can do to prevent these attacks. One is to block packets destined for the ‘‘r’’ commands from ever entering your network. The ‘‘r’’ servers listen at ports 512 (`rexec`, used by `rcp`), 513 (`rlogin`), and 514 (`rsh`). These are connection-oriented services, and use the TCP protocol. Blocking these with a packet filtering router is relatively easy to do. Firewalls based on proxy servers typically block these by default.

You can also defend individual systems by turning off these services if you don’t intend to use them. The `/etc/inetd.conf` file contains entries for each of these services (see Listing 2). You disable these services by putting a octothorp (pound sign or `#`) at the beginning of the line, and sending a hangup signal (SIGHUP, use `kill -HUP`) to the `inetd` process.

The ‘‘r’’ commands are certainly convenient. You can do tricks with them, for example, access a remote tape drive as if it were local by using `rsh` to drive `dd`.

But the lack of strong authentication is a terrible weakness. Newer versions of the ‘‘r’’ commands include support for Kerberos IV, which means if you are using Kerberos you could use the ‘‘r’’ commands in relative safety. If not, it is time to make some decisions about your site, or at least the systems under your control, about supporting these utilities. And check those Suns. Remember that any system you have upgraded to a later version of SunOS may have the plus sign in `/etc/hosts.equiv` (unless you overwrote it with your own file during installation.)

Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com

Go To Content   Search

Last Modified: Tuesday, 22-Aug-95 15:56:02 PDT