;login:

THE MAGAZINE OF USENIX & SAGE
August 2002 volume 27 • number 4

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

musings

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

rik@spirit.com

I often take a look at a past column before ramping up the old rant engine. Exactly a year ago, I was complaining about how a default install of Linux as workstation left your system wide open, with many unnecessary network services running. Now, I can cheerfully report that the opposite is true, that a more recent release of Linux (RedHat 7.2) not only left services disabled, but also included optional installation of netfilter (*http://netfilter.samba.org*), configured to protect your workstation. How nice.

Not that everything has come up roses. I also ranted (last year) about weaknesses in IE and XP. These issues have not gone away. IE still continues to be exploited on a regular basis. I have been getting several copies of the Klez worm, an Outlook virus, every day. And I port scanned a default install of Windows 2000 Professional Server and found that it had 25 TCP ports running services. This scan showed the simple services, good for denial of service, the ever present SMB share everything services, plus some new ones that are security related (Kerberos, ldapssl, and kpasswd5). Something shows up at port 6666/tcp, nominally IRC, but I don't believe that Win2K comes with IRC by default.

I have a useful security suggestion and a rant for this month as well. First the suggestion, then the rant.

Backfire

I spend a lot of time researching and thinking about attacks. The Honeynet Project (*http://project.honeynet.org*) is a really great source for learning about opportunistic attacks. Opportunistic attacks are those without any particular target in mind – any vulnerable system will do. These usually come about through scanners running on previously rooted boxes. The average length of time between when Project members install a default configured system and it gets rooted appears to be two days, maybe less. Something to think about, especially if you are in a network without any sort of firewall or controlled update program.

The other really interesting thing is what attackers do when they root a UNIX/Linux box. The standard sequence of events goes something like this:

1. Clean up signs of the attack.
2. Install new user accounts (one normal, one root).
3. Login via new user account, `su root`.
4. Download patches.
5. Download tools.
6. Install tools.
7. Logout.

The installation of patches really surprised me at first. Who would have thought that attackers would be so kindhearted and altruistic as to patch the security hole just used to root the box? The truth is much simpler – if the attacker got in this easily, so can anyone else, so the box must be patched at once. If they don't patch, someone else will soon "own" the system.

The tools downloaded vary a lot. Commonly, rootkits get downloaded. You can learn a lot about rootkits by visiting *http://www.chkrootkit.org/*, as well as by finding a script for detecting installed rootkits. Scanning and attack tools are also popular, turning the recently rooted system into yet another tool for global domination. Distributed denial of service (DDoS) agents or handlers may be installed. The system may also be used to run a bot, such as eggdrop (*http://eggdrop.org*), and to keep sysop privileges on some IRC channels.

What does all of this activity have in common? It all requires downloading software to the rooted system. Attackers most commonly use FTP, although they have been known to use TFTP (especially on Windows boxes), and could use other mechanisms, like Lynx (which version 1 of the lion worm did) or wget, as well. No matter what the attacker uses, downloading software requires an outgoing network connection.

If you have a public Web server sitting behind a firewall, you can easily prevent outgoing connections from that Web server. Web servers accept requests, they don't make outgoing connections to the Internet. A Web server could be performing DNS lookups, but this is generally not done, since it is slow and can be done offline when logs are analyzed (if anyone even cares about the FQDN of all visitors).

I suggest that you use firewall rules to block outgoing connections from public Web servers. Even better, set the firewall up so that you get paged when the Web server attempts to make an outgoing connection. At the very least, send yourself an email from the firewall. An outgoing connection should NEVER happen, so if it does, something very bad has happened.

I suggest that you use firewall rules to block outgoing connections from public Web servers.

This firewall rule also blocks scanning of Internet addresses from a public Web server, another common behavior seen from successfully attacked systems. Firewall rules like this would have done a lot to slow down the various versions of Code Red, but not Nimda, which used other attack vectors (JavaScript that executed the virus code, README.EML or README.EXE).

Using this simple trick with other public servers will not be quite as effective. SMTP servers make lots of connections to other SMTP servers, and if an SMTP server gets hit with an SMTP worm (how could anyone forget the Morris Worm?), these connections will also go to other SMTP servers, on port 25/TCP. Same thing with DNS servers, who naturally make connections to ports 53 TCP and UDP. Still, only allowing connections on these ports and sending out a page or email when something like FTP gets used from your public SMTP relay or DNS server would catch some attackers.

If you really want to go further, do this for your entire internal network. You cannot, practically speaking, only permit connections to specific remote hosts or services. You can, however, block access to services forbidden by policy, and use this to set off alarms. The popular Windows Trojan horse SubSeven (and others as well) will use a connection to an IRC channel, or ICQ, to announce that it has been successfully installed. If your policy forbids the use of IRC and/or ICQ, then block them, and alarm on them.

In May and June of 2002, two sites were quietly compromised, and the configuration scripts for software had a handful of lines added to them. In each case, the configuration script now compiles a program, executes it, and removes the source and executable while someone is running `./configure`. The short program makes an outgoing connection to a site, and if the connection succeeds, execs a shell at the victim's end. Think of this as a reverse telnet, but using (in this case) port 6667/tcp. The shell runs as the user running configure. A firewall rule that blocks IRC (which just happens to be the port used) would have blocked this backdoor. Too bad that one of the packages backdoored was the Bitchx IRC tool (meaning that the choice of the IRC port almost guarantees the port would be open).

Ever read the End User License Agreement that comes with any software or operating system you have ever acquired? Its only warranty is for the media it comes on.

Other great examples are the Microsoft everything sharing ports, 137/udp and 139/tcp. Certain attacks against IE encourage the victim to connect to a remote file share, which gets treated as part of the Local Zone for IE security purposes. Always block outgoing connections to port 139/tcp.

Marcus Ranum has talked about these types of tricks a lot in the past, and probably will in the near future (he is teaching a class about Honeypots at the USENIX Security Symposium). Ranum, in a past life, installed burglar alarms, and has talked about tricks like putting an alarm on a bogus jewelry box (one that the owner knows never to open). Having your firewall set off an alarm when something occurs that should never occur works as well, and perhaps better, since the alarm is on a separate system, not the one that has just been rooted.

DoR

A conservative think tank, the Alexis de Tocqueville Institution (ADTI), published a White Paper this summer entitled "Opening the Open Source Debate." The paper does not really debate anything but instead contends that open source "opens the gate to hackers and terrorists." I certainly consider this a bogus and misleading statement, and Microsoft has admitted that they give money to ADTI but will not say if they funded this report.

Jim Allchin, group vice president of Microsoft, in testimony in the MS antitrust case, had a lot to say about his own company's software (*http://www.eweek.com/article/0,3658,s%253D701%2526a%253D26875,00.asp*). Allchin said that "sharing information with competitors could damage national security and even threaten the U.S. war effort in Afghanistan." Wow. If Microsoft were to share their "secret," proprietary extensions to Kerberos Five, national security will be weakened? I don't think so.

The truth comes out a bit later in the same article. There Allchin acknowledged that some Microsoft code was so flawed it could not be safely disclosed. Included in this was the part of the Message Queuing protocol that contains a coding mistake which would threaten the security of enterprise systems using it if it were disclosed.

Well, that is certainly interesting news. Microsoft wants people to trust them and use their OS software for enterprise-level systems, yet knows it has fatal errors within it. If this were open source it would simply be patched, but because it is Microsoft it will remain hidden, at least until someone discovers and discloses it – perhaps with the Queuing worm? And which type of software are you going to trust now?

Andy Oram, of O'Reilly, has coined a term for this behavior. He called it "Denial of Responsibility (DoR)" in *http://www.oreillynet.com/cs/user/view/wlg/1500*. Oram's DoR attacks are about vendors shipping code that they know has killer bugs in it (see above). DoR is also about organizations and agencies that require the use of buggy software, so that information that should be protected gets exposed instead.

I'd like to take the concept further and consider DoR in terms of licensing. On the one hand, we have the GNU copyleft, open source, and the BSD license which describe the rights and privileges of the users of open source (or some related variant) software. Pundits argue that when open source software is used, no one is responsible for errors, omissions, or bugs in the code.

And what about proprietary software? Ever read the End User License Agreement that comes with any software or operating system you have ever acquired? Its *only* warranty is for the media it comes on. You use the software, it is your responsibility. Now, there is a real DoR attack. And, it is standard operating procedure today.

I like to compare the state of the current software industry to the automobile industry in the past. Do you know when automobile manufacturers started using safety glass in windshields? I really didn't appreciate this factoid until I watched a series on safety and liability issues on PBS. I also was made aware of the issue when my daughter told me that her husband was adding safety features, including modern brakes and safety glass, to the Model-A Ford he was turning into a hot rod.

You might even see Model-A Fords these days; they are really popular as restorations and hot-rod conversions. Their flat windshields are very distinctive and were once also lethal weapons. The ordinary plate glass used in the windshields could explode into razor sharp shards anytime a rock hit the windshield. It wasn't until the mid-'50s that safety glass became common in windshields. That's right. The cars your parents might have been driving (mine were) had plate glass windshields.

That was 50 years ago. Today, when Ford designs an SUV that rolls over quite easily, it has a big battle with a tire manufacturer, trying to place the blame for a dangerous design somewhere else. Newer Ford Explorers, however, have a wider wheelbase, making them less prone to rollovers (something pointed out to Ford many years ago). But do software vendors even bother? Nope. You use the software, therefore you are to blame.

With this level of responsibility, it is a wonder that *anyone* pays for software. Vendors take as much responsibility for their software, and the potential damage it might do, as does the open source community. If you are forced to use non-open source software, from a vendor who publicly declares that revealing information about the protocols and APIs involved is definitely dangerous, I think you are making a big mistake.

It's like driving a car with a plate glass windshield, just hoping that that truck in front of you doesn't toss a rock at you.

It is a wonder that *anyone* pays for software.