;login:

THE MAGAZINE OF USENIX & SAGE
February 2003 • volume 28 • number 1

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

It's the dead of winter, and global warming seems more like a mirage than the reality that it is. I can see snow on the Mogollon Rim, the southwest edge of the Colorado plateau, a welcome sign of moisture that will hopefully be the end of four years of drought.

On that cheery note, I get to muse about security, the topic that makes me pointy-headed. And a couple of things have got me riled up, both as a result of attending yet another security conference.

While teaching, I covered the topic of tunneling through firewalls. Tunneling has become very popular in open source of late, since four distributions had been trojaned as of the end of November 2002. In each case, someone broke into an FTP site and modified the configure script for a particular package so that it would compile and execute an additional program. This program, the trojan, runs in the background with the privileges of the user who executed configure – another reminder of why you avoid doing anything other than what you must do as root. And what the trojan program does is pretty cute.

Once an hour, the trojan initiates an outgoing connection to a fixed IP address, and port 6667/TCP. If the connection is successful, it only remains open for 10 seconds, unless some input is received. When something is received, the trojan executes a shell. I suppose that the attackers had arranged something akin to an expect script at the remote end of this connection, which will download assorted tools and kits, and automatically continue to exploit the victim of the trojan. Of course, as the trojan used fixed IP addresses (different for each instance), as soon as someone noticed the attack, the site receiving the connections could be taken off the Net and cleaned up.

Once upon a time, I used to counsel people to examine the source code that they download from the Internet before using it. Today, that admonition is incredibly absurd, especially when you can download entire CDs containing operating systems along with software packages numbering in the tens of thousands. I recently grabbed a Linux rootkit, to use in a class example, and realized that checking it out for *unexpected* back doors (I knew about the well-known ones) was going to take longer than I cared to spend. There are shortcuts, such as looking for system calls that open files, create sockets, and execute programs. The trojan mentioned above both creates sockets and executes a shell, so it qualifies. But who even considers checking for these things? Keep in mind that many programs do this legitimately. The configure script found in cfengine (not one of the victims) contains 48 lines that include `socket`. Checking the GPG signature of packages at least assures that nothing has been added since the package was signed.

Sysadmins working at sites with serious firewalls can consider blocking all outgoing traffic except that which is permitted and expected. Blocking arbitrary outgoing traffic, and watching your logs for any deny messages actually will help you discover exploited systems on your internal networks.

That is, unless the attacker is using HTTP over port 80/TCP. If someone out there works at a site connected to the Internet that does not permit access to most Web servers, please let me know. I include the word "most" for those sites unlucky enough to be behind firewalls that block requests for URIs by using "evil"-word or other filtering mechanisms. I really wonder how well such things can work, after winding up at a pornography page that had replaced Lord Somer's Lurker rootkit site.

If the writer of the configure trojan had decided to make his (or her) program about twice as long, it could have used port 80/TCP and included fake headers on the requests, and stripped off the server headers from the replies. Doing so might have made the trojan easier to spot (it is actually well disguised, complete with comments that help it blend into any configure script). But it would have made it much more difficult to prevent it from making outgoing connections through firewalls that permit HTTP.

And it is not only this trojan that uses this technique. An IIS 5 exploit named "jill" also made outgoing connections but, in this case, using the IP address and port of the attacker's choice. You should consider exploits that make outgoing connections something that you should expect, and not something rare and unusual.

SOAP

Exploits won't be the only thing slipping through your firewall soon. SOAP, the Simple Object Access Protocol (*http://soap.weblogs.com/*), also uses HTTP, with the actual goal being able to penetrate firewalls at will. SOAP carries XML payloads used to invoke remote methods, carry executable code, and return responses. Now, you can already invoke remote methods using CGI, so that is really nothing new. And returning results, ho-hum. But carrying remote code, well now, I hope that got your attention.

SOAP has become a carrier for .NET, Microsoft's new programming paradigm. DCOM, the old paradigm, did support the downloading of remote code and its execution using plain old HTTP. DCOM has, as one of its drawbacks, no means for running remote code securely. The .NET framework gets around this by permitting the programmer to request only the privileges needed to execute on the victim's, er, remote user's system. And the CLR (Common Language Runtime) that will execute the MSIL (Microsoft Intermediate Language) module that has been downloaded can also set limits on the privileges allowed the code, based on the source of the code and its authenticating signature.

.NET application security depends entirely on the security skills of the programmer writing the application server or remote modules.

VB.Net and C# both produce MSIL, something that is vaguely like Java bytecode. But, MSIL does not include the same security paradigms as Java, which means that programmers can still make mistakes that would be impossible using Java bytecode. Also, there is no security provided by SOAP or XML – they just carry the code. In other words, .NET application security depends entirely on the security skills of the programmer writing the application server or remote modules. Does this sound familiar?

If you want to get an idea about how things can go wrong, just check out MS-02-065. All MDAC (an ActiveX module used by IIS and IE) versions until 2.7 (the one installed on XP) have a buffer overflow that can be exploited using Web accesses (if RDS is enabled), HTML, and email that includes HTML. The MDAC ActiveX module has been digitally signed by Microsoft, so even if you never had it, or had removed it, a malicious email could be used to load a vulnerable version, and it would be trusted – because it was signed by Microsoft. You could actually disable Active Scripting (a very good idea, suggested by Microsoft), as well as remove Microsoft from the list of trusted providers of code (*http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-065.asp*).

SOAP appears to be an enabling technology for security holes. It supports transmission of mobile code while making it very difficult to check and see what is being sent – that information is buried in the XML. It really is a shame that security isn't the first consideration when new protocols are designed, instead of an afterthought. And on that note, I wish you a warm good night.