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

In my last column, I waxed enthusiastic about how superior Windows was when it came to the ease of writing keystroke sniffers. While there are hundreds more keystroke sniffers for Windows, I have since learned of a new one that runs on Linux (2.2 and 2.4 kernels), Solaris, and OpenBSD.

I was talking to Ed Balas, after being grabbed by Lance Spitzer and dragged over to a table staffed by the Honeynet Project at the 2003 BlackHat conference in Las Vegas. Ed explained to me that he was a "generalist, not a specialist," and had just finished writing, then porting, a kernel module that is designed for use in GenII Honeynet, which he is writing about for the December 2003 annual security issue of *;login:*.

The original honeynets relied on packet sniffing, and that can be a problem when the first thing the attacker does is download and install a version of SSH. Now, everything that gets downloaded can travel across an encrypted link. And attackers have started to encrypt their tools on UNIX systems, something that has been done for a while with Windows trojans and viruses, so static analysis becomes much more difficult. But if the honeypot can capture all keystrokes, the potential for useful research becomes much greater for the honeypot owner.

Sebec2 (see *http://project.honeynet.org/papers/honeynet/tools/index.html*) hooks into the read entry in the kernel's system call table, so that *anything* that a program reads, whether it be a file or a command line, gets captured. That means that any encryption keys or passwords that an attacker uses will be recorded. Sebec2 then creates its own IP packets and writes them directly to the NIC device driver. Most applications that create their own packets (e.g., hping) write to a raw socket, and that makes it possible for snort or tcpdump to capture the packet before it gets written to the device driver. But Sebec2 prevents the packets from being detected by any application-level process – in fact, by anything other that a kernel module like Sebec2 itself. The server end of Sebec2 still sniffs the network to collect the packets.

As Ed described his new tool, I couldn't help but think that this would make a great rootkit. Sebec2, unlike xkey (see my August 2003 *Musings*), also reports the application that requested the read(), so an eavesdropper gets a much clearer picture than with xkey. And Sebec is listed as a rootkit at chkrootkit.org. Like many other security tools, the potential is there for both good and dark uses.

I also heard about a really interesting study done by Cisco employees. They decided to take a closer look at the myths surrounding BGP vulnerabilities. When someone claims to be able to "bring down the Internet in a half hour," they would be referring to flaws either in the routers or in the core routing infrastructure, and that means BGP. Matthew Franz and Sean Convery first presented their results (publicly) at NANOG and subsequently at the BlackHat conference. You can read their presentation at *http://www.nanog.org/mtg-0306/pdf/franz.pdf*.

I wrote about the potential for "bad things happening" to BGP back in April of 2002, and nothing has changed since then. True, there are now two proposals for adding security to BGP updates, when before there was only one. But S-BGP (Secure-BGP) requires a PKI, the signing of routes by the originating router, and the checking of signatures by the router receiving the update: *http://www.ietf.org/internet-drafts/draft-clynn-s-bgp-protocol-01.txt*.

I heard complaints about this concept from many people, who pointed out that handling public key encryption was not something router CPUs could manage while still

getting useful work done. The S-BGP proponents do not consider this an issue, but if you do use BGP, and S-BGP becomes mandatory, plan on updating the core of your router so it will include the extra processing power necessary. The router company version is called SoBGP, or Secure Origin BGP, and differs in that it uses a decentralized certificate management system and relies on RADIUS servers (or something that does *not* involve the router's CPU) to handle public key encryption: *http://www.ietf.org/internet-drafts/draft-ng-sobgp-bgp-extensions-01.txt*.

Convery and Franz decided to take a hard look at the FUD surrounding BGP by creating attack trees, then trying out each branch to see if any of these attacks was actually something to worry about. Recall, if you would, that BGP neighbors, or peers, keep a TCP connection going at all times. Once a minute, they exchange keep-alive messages, or they might send routing updates to their neighbors. If this TCP connection (to port 179) goes down, each router strives to recreate it quickly while also sending out updates reporting that the routes they know of that travel through the neighboring router are all invalid. That information percolates across the Internet, forcing any router with a complete set of BGP routes to recalculate its routing tables. In Convery and Franz's attack tree, this leads to two branches: disrupt the TCP connection or insert phony routes (you can read the papers or presentation for much more detail).

So they tried disrupting the TCP connections between seven different products/implementations that all support BGP. And failed. They also tried hijacking BGP connections. BGP neighbors can use MD5 authentication that gets included as a TCP option, and if this is used, hijacking means guessing the shared secret. But hijacking also means being able to sniff communication between BGP neighbors, which are often connected using a link dedicated to that purpose. The short answer is that hijacking a connection could be done if MD5 authentication is not used, and the connection can be sniffed.

They also attempted inserting an update when MD5 was not used, and this resulted in an ACK storm (since the receiving system has sent an acknowledgment for a packet that the sending system has never heard of), with the eventual result that the connection gets reset after five minutes. But during that five minutes, the phony routes spread through the Internet (unless blocked by BGP route filtering, but that's another story).

They tried fuzz testing using BGP messages in attempts to crash BGP; four flaws were found in 1200 attempts, and three of those worked only if received from a valid peer and/or valid AS. The fuzz testing included sending messages that varied from totally random data to up to eight valid fields followed by random data. So the implementations appeared to be fairly robust.

The most interesting test that Convery and Franz carried out was to use traceroute to 120,000 destinations, one for each CIDR block in the Internet, in an attempt to map out all the routers that might be BGP speakers. With a list of 115,466 IP addresses ready, they then tried SYN probes against each address, to ports 22, 23, 80, and 179 (SSH, Telnet, HTTP, and BGP). In theory, any access to administrative ports by "outsiders" should be filtered or ignored. In practice, 14.5% of all routers responded to at least one of the three administrative ports, SSH, Telnet, or HTTP.

They concluded, based on other tests not mentioned here, that the best way to disrupt the Internet is to take over one or more routers and have them distribute misleading routes. If a trusted router gets compromised, signed updates (if these are eventually approved) would still be trusted. Misconfigured routers are, they concluded, a much bigger issue at present than problems with BGP.

Misconfigured routers are . . . a much bigger issue at present than problems with BGP.

Each vulnerability stretches way back in time, to much earlier versions, yet is still present in the Windows Server 2003.

July 2003 also brought with it announcements of several new buffer overflow vulnerabilities in Windows. One announcement affected all versions of Windows from 95 up to Server 2003 (CA-2003-14 or MS03-023) and the module that handles Rich Text within HTML. The other involved only NT through Windows Server 2003, and requires access to port 135 TCP or UDP (CA-2003-16/19 or MS03-026). Windows Server 2003 is supposed to be the most secure version of Windows, so the existence of these vulnerabilities was considered an embarrassment for Microsoft. But I noticed something other than simple embarrassment in these vulnerabilities.

Each vulnerability stretches way back in time, to much earlier versions, yet is still present in the Windows Server 2003. What that tells me is that Windows Server 2003 is still using code left over from the earliest days of Windows with a TCP/IP stack (the RTF-HTML bug) or from the earliest days of NT. I have always thought one of the biggest issues with Windows systems is the amount of legacy code each version contains, crucial to maintaining backwards compatibility. Microsoft built all of Server 2003 with stack canaries, similar to the StackGuard canaries for Linux, but these servers are still vulnerable.

One reason is that stack canaries can only guard against stack overflows, not heap overflows. An even better reason is that some mistakes in code are so subtle that code analysis by programmers and by special tools will not catch them. The heap overflow found in Sendmail in March of 2003 was a good example, as understanding the source code meant single-stepping through it, a laborious process.

I got a chance to read most of *Secure Coding*, a new O'Reilly book by Mark Graff and Ken van Wyk, both currently with CERT. Secure Coding is about principles and practices, and it provides little in terms of code examples (and most examples are the mistakes, not solutions). Their goal is not to write a programming guide, since other people have done that well, but instead to share their years of experience with vulnerabilities. Problems offered go beyond programming mistakes, such as the all-too-frequent buffer overflows, to design and operational mistakes.

They do include some great examples of failures and of good design. I liked their explanation of how one could log in to some rlogin servers using the login name -froot and get a root prompt with no password (a problem that occurred because rlogind calls the login program without checking the provided login name). Or how /etc/passwd file fragments started showing up in Solaris tar files in a new release. It turned out that some heap memory had just been freed, and then reallocated as the buffer used when writing out tape blocks, and that memory had previously been used in password file reading functions (getpwent()). Both of these issues are problems in composition – that is, neither occurred on its own, but required a combination or sequence of events for the bug to assert itself. In the case of tar, some code had been removed, moving the freeing of the memory used by getpwent() closer to the malloc() used for tar's block buffers. Who would have ever guessed (much less figured this out) before it was discovered the hard way.

Secure Coding is a relatively short book (177 pages), but not an easy one. It really helped me to understand why I was never a great programmer – my programming style was more prototyping than design. Prototyping was loads of fun, as I would occasionally knock together example programs thousands of lines long in a day when I was teaching people how to use a graphics library. I was proud that I could get such a monstrosity to compile, link, and actually demonstrate potential solutions to the problems the client had. But the real problem arose after I left, when the client decided not to

toss those thousands of lines of disorganized code, but instead to use it as a starting point for their application. Note that what I was doing was gluing together many short example applications, not writing fresh (but unorganized) code.

Graff and van Wyk provide methodologies and questions that you can use in architecture, design, implementation, and operation. Having read their book, I finally recognized that secure coding cannot be taught as an isolated subject but needs to be woven into every area of software instruction. I certainly recommend this book if you have anything to do with software (beyond just running off-the-shelf apps), as security is important and not something that can be taken lightly (or worse, something that you can just expect will happen organically, without careful thought).

I finally recognized that secure coding cannot be taught as an isolated subject but needs to be woven into every area of software instruction.

As I finish writing this column, I am also getting ready to leave for the USENIX Security Symposium in Washington, DC, something I have been looking forward to for months. BlackHat is interesting, and some research does show up there, such as the Cisco presentation. BlackHat led off this year with a presentation by David Litchfield, who, in his pre-conference notes, planned on explaining the differences between exploits in Linux and Windows using stack-based buffer overflows in Oracle 9i's XDB as the example. Instead, he chose to talk about the much more current issue of the overflows in Windows RPC.

Litchfield certainly knows his stuff, that being the internals of Windows and Linux executables and libraries. Although his presentation was disjointed, he did manage to take the audience through a code debugging session of MS RPC, showing where the problem occurs and how it can be exploited. He could not demonstrate an exploit for Server 2003, which LSD claims to have written but not released (yet). But he did have parts of his audience absolutely mesmerized.

A group of Microsoft employees were sitting in the front of the room, furiously taking notes. Litchfield described how the stack overflow protection works in Server 2003 and suggested methods for evading it, something LSD appears to have done.

This vulnerability in MS server products has great potential for worm writers, as the MS-Blaster worm and its variants have shown. Microsoft, having been slammed by Slammer, had adopted a much less tolerant view of people who fail to patch their systems. Everyone within Microsoft was given a deadline to install the patch for MS03-026. If they failed to do so by the deadline, their network connection would be disabled. I asked a friendly Microsoft employee exactly what that meant, and he told me that internal security was aggressively scanning for the vulnerability, and would turn off the switch port of anyone not patched by the deadline. Also, anyone logging in remotely would also be tested before their authentication could be completed. Wise moves. Apparently, Microsoft's customers did not take the issue seriously enough, as MS-Blaster has been moderately successful, although paling in comparison with the SoBig.F virus. When will they ever learn?