FEATURES

March 2001

OPEN-SOURCE SECURITY

OPEN SOURCE UNDER THE HOOD

Vendors are increasingly including open-source components in their commercial products. What impact does this trend have on product security?

BY PETE LOSHIN

The days are long gone when all you needed to start your own software company were a compiler and a computer. Creating commercial off-the-shelf (COTS) products from scratch in today's market is a daunting task for any but the biggest software companies. Smaller vendors have to compete with the likes of Microsoft, Sun and Cisco, with only a fraction of the resources.

Almost no one can afford to build their own new products from scratch anymore, and the problem is magnified for vendors of network appliances: They've got to deliver a functional, competitively priced server, including software and hardware, while still turning a profit. Vendors of other products, from operating systems to software suites to end-user workstations, are feeling the pinch as well.

Considering this environment, it's not surprising to find vendors increasingly turning to open-source code when creating new products. Yet buyers may not always be aware that inside their shiny new firewall lurks an open-source OS, such as Linux or FreeBSD. Network security appliances designed to do firewalling, intrusion detection and other security functions often rely extensively on open-source OSes and utilities. But many other products include open-source components as well. Apple's new Macintosh OS X, for instance, is based on Free BSD 3.2 and the Mach 3.0 project from Carnegie Mellon University. Apache, BIND, Sendmail and Perl are all widely used in both commercial and non-commercial products.

Among the obvious reasons developers turn to open source are cost and security. Clearly, vendors can keep their costs down when they don't have to build their own components or buy licenses for commercial components. Why build a Web server when you can use the best one around-Apache-for nothing? Why build your own OS when you can use FreeBSD? Why not include open-source security utilities with a commercial security product?

While some people automatically assume that open-source OSes are more secure than proprietary OSes, it entirely depends on how the code is used and supported. When done right, open-source components add real value to commercial products-and are likely to be at least as secure as closed-source components.

So what exactly is open-source code, and what impact does it have on product security? How can it affect your systems and networks? How can you tell if the product you're using incorporates open source? And how can you become an intelligent consumer of products that use open source?

Open Source, Everywhere

According to the Open Source Initiative (OSI) (www.opensource.org), nine attributes define what can be considered open-source code (see box below). Among these stipulations, the most important one is that the original source code itself must be available to users, but open-source licenses can vary in terms of how much they restrict use of the code.

"Vendor-friendly" open-source licenses-such as those used by BSD-based projects[1] and the Apache Software Foundation-make it easy to adapt an open-source program into a profitable commercial product. The BSD and Apache licenses allow anyone to modify and distribute programs as closed-source products, as long as appropriate copyright notices are included. Some, like the Apache license, forbid use of the open-source project's name in any commercial version without written permission from the project.

On the other hand, the GNU Public License (GPL) is much stricter. The GPL permits the sale of copies of GPLed software, placing no limits on the prices charged. However, if only binaries are provided, the vendor must make the source code available free. More to the point: The GPL requires anyone who modifies and distributes GPLed code to make the modified source code available as well. You are welcome to modify the code, but your modifications must also be entirely free and available to anyone who uses them.

The nice thing about open-source software is that so much of it is released under relatively open licenses. Even where there are limits on how you use and distribute the code, developers have remarkable freedom in using programs they would otherwise have to develop on their own or license from other vendors-both of which involve more time, expense and development effort.

The problem with open source in commercial products is that it's "largely invisible," says Jon Lasser, lead coordinator for the Bastille Hardening System (http://bastille-linux.sourceforge.net), a script that tightens up the security of systems running stock Red Hat and Mandrake Linux. Lasser points out that "Solaris-and just about any commercial Unix I can name-has included Sendmail for more than a decade," adding that most Unix-based nameservers also run the open-source BIND code.

[1] Berkeley Software Distribution (BSD) is the "freed" version of AT&T's original Unix operating system.

Where to Look, and What You'll Find

Open-source components can show up almost anywhere, though they're most prevalent in network appliances-devices designed to provide plug 'n' play network services. Network appliances must communicate over a network as well as provide networking services: file and printer services, e-mail, Web and Web caching, file transfer, terminal services and so on.

Network security appliances generally offer firewalling and/or virtual private networking, and perhaps other services such as single sign-on (SSO) or content filtering. Any or all of these functions are easily provided with almost any open-source BSD-based OS or Linux. For example, Cobalt Networks (www.cobalt.com) uses Red Hat Linux as its base OS. WireX Communications (www.wirex.com) adapts and hardens Red Hat Linux and then packages it into a software network appliance that can be licensed for resale (see Table 1 for other examples).

But appliances aren't the only products in which you'll find open-source components. Others adapt the open-source code to create their own proprietary products. C2Net Software (www.c2net.com), for instance, created its commercial Stronghold Secure Web Server with Apache and OpenSSL. Other examples include Send-mail or the SAINT vulnerability scanner, which you can either use free under open-source licenses or buy in enhanced commercial form from Sendmail Inc. (www.sendmail.com) and WWWDSI (www.wwdsi.com), respectively.

There are also programs that support open protocols, which may or may not be open software. For example, SSH is a protocol for secure shell sessions over a network; it's also a proprietary product from SSH Communication Security (www.ssh.com). Several other SSH implementations are also available, both proprietary and open source.

Then there are the "formerly open-source" products such as the Tripwire intrusion detection system (IDS), originally developed as free software but later re-engineered from scratch as a closed-source program licensed by Tripwire Inc. (www.tripwire.com). Though the original Unix-only free version remains free, the proprietary product is also available for NT, with more features and better support.

In the Unix world, quite a few key programs (particularly networking software) are open source. BIND is almost universally used for Domain Name System (DNS). Then there's the Perl scripting language, frequently used for network and server administration on Unix, Windows and other platforms. Vendors may offer programs that require open-source system tools. For example, Okiok Data (www.okiok.com) offers S-Filer, a tool that uses Perl for secure file transfers.

Other vendors, especially those providing customized programming, use open source in their work products. For example, ArsDigita Corp. (www.arsdigita.com) builds and distributes the ArsDigita Community System (ACS), an open-source application development platform and suite of enterprise applications. Many other vendors support open-source projects, often hiring programmers to work exclusively on those projects.

Risks and Rewards of Open Source in COTS Products

The common assumption among developers and engineers is that the "many eyes" approach

to open-source code projects makes it inherently more reliable, robust and secure than closed-source code. However, recent revelations of glaring holes and vulnerabilities in sometimes quite old and widely used open-source code have led some to question the validity of this assumption. Steve Lipner, manager of the Microsoft Security Response Center, points to the recent discovery of vulnerabilities in MIT's Kerberos network authentication software, "where buffer overruns went undiscovered for nearly a decade" in the widely distributed and implemented open-source code. (Microsoft released a closed-source-and slightly non-standard, and thus non-interoperable-version of Kerberos with its Windows 2000 suite.)

While few open-source advocates still claim absolute security superiority over closed source, most experts seem to agree that open source has the potential to be at least as secure, if not more secure, than closed source. For one thing, open-source code by itself should have no real adverse effect on system security. "The negative ramification that people cite is that -anyone can look at the code to find holes,'" says Bastille's Jon Lasser. But that works both ways, since open-source vulnerabilities are (in theory anyway) vetted by a much larger community of developers and engineers.

"I'm not convinced that there are any significant real advantages to going with closed source unless there is something about the security mechanism itself that intrinsically can't stand up to examination" says Paul Robichaux, a senior solutions architect for EntireNet and author of several books on Microsoft products.

Robichaux stops short of unqualified endorsement of the open-source security model, cautioning that having "more eyeballs looking at [open-source code] is no guarantee of quality." Moreover, for some systems, such as national security programs, close public scrutiny is unwarranted. "If you look at the authentication system...the [U.S.] National Command Authority uses when they want to tell somebody to launch a nuclear missile, you probably would not gain any security from having many more eyeballs looking at that."

Microsoft's Lipner acknowledges that "no software is free from flaw," while suggesting that "the difference between products lies in how actively vendors seek out the flaws and then fix them." According to Lipner, Microsoft "pays top dollar to ensure that its software is scrutinized by the best minds in the industry rather than taking the open-source approach of relying on hobbyists and--someone else' to scan code in their spare time."

Microsoft's official position on the relative security of closed- and open-source software, according to Lipner, is that "the difference lies in how we-do' security. The most fundamental question to ask when examining the security of any software is whether or not the design and development process results in a sound and secure design and a solid implementation." Microsoft isn't opposed to open reviews of cryptographic protocols and algorithms; Lipner says that they "can definitely improve security. These are generally simple enough that academic or external review can find issues and add value."

However, Lipner points out that securing large software systems calls for "a substantial and often costly level of resources applied by a full-time team"--which, he says, will only work if the costs can eventually be recovered by product revenue.

According to Bastille's Lasser, "Anyone using open source can fix the code, or pay someone else to fix it. And anyone can examine it." One result of this all-hands-on-deck model is the potential for discovering backdoors. For instance, Lasser points to the Borland InterBase, initially a proprietary product that, after seven years, was released as open-source code. The proprietary version contained a backdoor that wasn't discovered until six months after the code was opened.

Kurt Seifried, senior analyst for SecurityPortal.com and project head for the Linux Security Knowledge Base, explains that attackers don't need access to the source code to find and exploit problems. For instance, Microsoft issued more than 100 security advisories in 2000, and new bugs related to IIS or IE are publicized on Bugtraq and NTBugtraq all the time.

That's not to say that open-source code has no downside, particularly when implemented in commercial products. It all depends on who is doing the implementation and how support is being provided. Lasser suggests that in order to keep customers' code current, vendors should offer opt-in e-mail lists for up-to-date news about the product, be up-front about any security problems and provide patches online that have been cryptographically signed. While vendors such as Red Hat, SuSE and Mandrake offer these services, "few vendors do all of this," he says.

Seifried, singling out OpenBSD, says some open-source projects are particularly well suited to secure deployment in commercial products. "They sat down and spent a large number of man-years auditing it heavily and now have a pretty solid and secure codebase to work from." Many commercial vendors use OpenBSD--as well as FreeBSD and NetBSD--for firewalls.

However, inappropriately using a secure and open program is dangerous. "It's almost always a question of how the products are used, rather than what they are," Lasser notes. "One of the defining characteristics of the security problem is that these evaluations are fluid, depending largely on new exploits and classes of exploits that are discovered." Just because OpenBSD is noteably secure doesn't mean it's not still vulnerable to common exploits of programs like FTP, DHCP and Send-mail. If someone used OpenBSD as part of a "secure FTP solution," but used an insecure FTP implementation, "they're toast," says Lasser.

Buying Open Source Under the Covers, Intelligently

Vendors incorporate open-source code in their products differently, so simply scrutinizing brochures or Web sites isn't enough. Some vendors make open source an important part of their marketing strategy, pointing to it as a source of strength. Examples include C2Net and Linux-based firewall vendor Cybernet Systems Corp. (www.cybernet.com).

Network security appliance vendors sometimes include lists of software installed on their hardware. Read the fine print in datasheets for Sun's Cobalt RaQ, Qube and other network appliances (www.sun.com), and you'll see that those products use the Linux 2.2 kernel. Axent's (now Symantec's) Raptor firewall appliance (www.symantec.com) is also based on the Cobalt RaQ. Other vendors are less forthcoming, releasing appliances based on "proprietary" OSes that are, in fact, open-source based. For example, the FireBox network appliance from NetWolves (www.netwolves.com) is based on FreeBSD-but the company's Web site refers to it as a "Unix-based FoxOS" operating system.

The greatest benefit of buying products that incorporate open source can also be part of the greatest drawback--that is, the fact that vulnerabilities and exploits for leading open-source products are widely published. This means fixes are usually made available quickly, but it also means that if you take too long to update your systems, they will be vulnerable to script-kiddiez and other attackers.

Bastille's Lasser suggests that vendors should provide proactive support, notifying customers of vulnerabilities and fixes. However, in his opinion, knowing where a particular piece of a system originated, whether open source or not, is not always very useful. "Sure, you could ask whose TCP/IP stack they used, but you won't know which version, and the optimal solution varies by week, application and phase of the moon," Lasser opines. But he also acknowledges that "there's nothing especially specific to open source about any of this."

In the final analysis, it's up to consumers to keep track of what open-source code is running on their systems--if only to keep them up to date. SecurityPortal's Seifried suggests asking vendors for a list of their product's security patches. "If they don't have any patches, I wouldn't buy it. Nothing is perfect.

"Open source is like any technology," he adds. "The implementation can be good or bad. Vendors that use open source and issue timely updates, proactively audit code and so on are good; vendors that don't should be avoided if possible."

Could You Do It Yourself?

Vendors use open-source code to build their products, so why can't anyone else? Well, nothing's stopping them, but the question is whether they can afford to (see box, below). A vendor can afford to put significant resources into putting together a package from open sources as long as they anticipate revenues. Seifried says it's a matter of convenience. "I can easily download the Linux kernel source and all the source code for software I need. Turning that into a working e-mail server, on the other hand, is a completely different matter."

According to Lasser, there are three other good reasons not to "roll your own." First, commercial versions of open-source programs usually incorporate proprietary extensions that add significant value. For example, C2Net's Stronghold Web server adds strong encryption and other features to Apache. Also, with proprietary products you get the benefit of quality assurance. And finally, you get support. "You're not paying for the software so much as you are to have someone to complain to when things break," Lasser says.

Buying commercial products based on open-source components may give users the best of both worlds. Microsoft's Lipner suggests that "proprietary systems are better reviewed, better tested and have a more robust process for dealing with security vulnerabilities when they are found"-though he was thinking more of entirely proprietary systems like those available from Microsoft. Everyone seems to agree that a proprietary product provides greater ease of use, better support, more convenience and more features, whether or not the proprietary product incorporates open-source code.

THE NINE ATTRIBUTES OF OPEN SOURCE

1. Free Redistribution
The license may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

2. Source Code
The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost-preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works
The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of the Author's Source Code
The license may restrict source code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups
The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor
The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License
The rights attached to the program must apply to all to whom the program is redistributed, without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product
The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Contaminate Other Software
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

Source: Open Source Initiative, www.opensource.org

OPEN SOURCE INSIDE

Download pdf

BACKDOORS: OPEN OR CLOSED?

Backdoors are the security manager's nightmare. About the worst thing that could happen from a security standpoint would be the deployment of a system that includes an unknown backdoor.

The fear of backdoors "has probably been the biggest drag on the adoption of open source in the commercial world," says security expert and author Paul Robichaux. He recommends taking great care in reviewing any code brought in-house. "If you're using open-source code and you're not already reviewing it very carefully, you're being stupid and you deserve what you get," he says.

In the open-source world, it's likely that any externally injected malware (such as a Trojan) will be caught before it can be incorporated into production systems. But Robichaux warns that when you are buying compiled products (such as security appliances)--whether open source or not--"you never know what's going to be in those."

Rumors of backdoors inserted into commercial products have persisted for years. According to Robichaux, it's plausible (though never confirmed) that government agencies such as the National Security Agency (NSA) could "go to Microsoft or Sun or Oracle or whoever and wave their magic national security wand." As a result, "The product you're using will have a hole in it, but you won't necessarily find out."

Those using open-source code-whether it's the native code itself or a COTS product based on it-face a Catch-22 when it comes to backdoors. On the one hand, attackers are more likely to try to insert a backdoor into open-source code because, unlike closed source, it's out in the public domain for everyone to play with. On the other hand, they will be less likely to succeed because there are so many other people, with different goals, looking at the same code, which increases the possibility that it will be noticed.

BUILDING YOUR OWN FIREWALL

Time is money, and it's well worth spending a few thousand dollars to save a few weeks of a security manager's time.

When the Internet was still a research network, it was built on BSD/Unix systems. BSD derivatives, like all Unix flavors, are designed from the ground up to run on networked devices. So it shouldn't surprise anyone that so many firewalls and Internet servers are based on BSD-related distributions. Linux, with its relatively easy-to-use firewalling and Network Address Translation (NAT) functions, is also a popular platform for security applications.

If you have Linux, BSD or Unix expertise--or at least plenty of time--BSD- and Linux-based firewalls can be cheap and effective security solutions. But doing it yourself can be an invitation to disaster unless you're sure you've done everything right.

Once you decide to build your own firewall, you must install the operating system as securely as possible, and then create firewall rules to keep out all unauthorized traffic. That means building security policies first--a prerequisite for any firewall.

First, you must choose the most appropriate OS. Some prefer Linux for ease of use and widespread support; others find one of the BSD flavors (OpenBSD, NetBSD, FreeBSD, etc.) stronger (though perhaps less user friendly). If you choose Linux, you now have the option of using the 2.2 kernel, which provides firewall support with packet filtering by the ipchains program; or the 2.4 kernel, which uses the iptables program to create stateful inspection firewalls.

The next step is to get a trustworthy distribution: that means downloading from a trusted Web site or buying it on CD-ROM-and checking the distribution's digital signature. You'll want to review the source code before compiling it, and you should compile the kernel with only the drivers that are absolutely necessary.

Once installed, you'll need to turn off all extraneous services and harden the operating system in other ways (see Resources). You can do it by hand, or use a Linux-hardener (for example, the Bastille hardening scripts). Then, you've got to develop your firewall rules: what kind of packets should be filtered-both inbound and outbound-what applications are permitted, and so on.

Building and configuring the box, of course, is only the first step. Once the firewall is in operation, you must constantly monitor logs for suspicious activities, watch out for security alerts and install security patches as soon as they are available.

Some of these tasks (setting firewall rules and staying on top of security alerts, for example) are necessary with any firewall. But if you roll your own, you don't have the option of outsourcing any of them to a commercial firewall vendor.

RESOURCES

Download pdf

Columnist PETE LOSHIN (pete@loshin.com) is a senior editor-at-large for Information Security. He produces the Internet-Standard.com Web site and has authored more than 20 books on Internet protocols and security.

HOME