Bruce Schneier

Crypto-Gram Newsletter

November 15, 2001

by Bruce Schneier
Founder and CTO
Counterpane Internet Security, Inc.
schneier@counterpane.com
<http://www.counterpane.com>

A free monthly newsletter providing summaries, analyses, insights, and commentaries on computer security and cryptography.

Back issues are available at <http://www.schneier.com/crypto-gram.html>. To subscribe, visit <http://www.schneier.com/crypto-gram.html> or send a blank message to crypto-gram-subscribe@chaparraltree.com.

In this issue:

Full Disclosure

Microsoft is leading the charge to restrict the free flow of computer security vulnerabilities. Last month Scott Culp, manager of the security response center at Microsoft, published an essay describing the current practice of publishing security vulnerabilities to be "information anarchy." He claimed that we'd all be a lot safer if researchers would keep details about vulnerabilities to themselves, and stop arming hackers with offensive tools. Last week, at Microsoft's Trusted Computing Forum, Culp announced a new coalition to put these ideas into practice.

This is the classic "bug secrecy vs. full disclosure" debate. I've written about it previously in Crypto-Gram; others have written about it as well. It's a complicated issue with subtle implications all over computer security, and it's one worth discussing again.

The Window of Exposure

I coined a term called the "Window of Exposure" to explain the evolution of a security vulnerability over time. A vulnerability is a bug; it's a programming mistake made by a programmer during the product's development and not caught during testing. It's an opening that someone can abuse to break into the computer or do something normally prohibited.

Assume there's a vulnerability in a product and no one knows about it. There is little danger, because no one knows to exploit the vulnerability. This vulnerability can lie undiscovered for a short time -- Windows XP vulnerabilities were discovered before the product was released -- or for years. Eventually, someone discovers the vulnerability. Maybe it's a good guy who tells the developer. Maybe it's a bad guy who exploits the vulnerability to break into systems. Maybe it's a guy who tells no one, and then someone else discovers it a few months later. In any case, once someone knows about the vulnerability, the danger increases.

Eventually, news of the vulnerability spreads. Maybe it spreads amongst the security community. Maybe it spreads amongst the

hacker underground. The danger increases as more people learn about the vulnerability. At some point, the vulnerability is announced. Maybe it's announced on Bugtraq or another vulnerability Web site. Maybe it's announced by the security researcher in a press release, or by CERT, or by the software developer. Maybe it's announced on a hacker bulletin board. But once it's announced, the danger increases even more because more people know about it.

Then, someone writes an exploit: an automatic tool that exercises the vulnerability. This is an inflection point, and one that doesn't have a real-world analog for two reasons. One, software has the ability to separate skill from ability. Once a tool is written, anyone can exploit the vulnerability, regardless of his skill or understanding. And two, this tool can be distributed widely for zero cost, thereby giving everybody who wants it the ability. This is where "script kiddies" come into play: people who use automatic attack tools to break into systems. Once a tool is written, the danger increases by orders of magnitude.

Then, the software developer issues a patch. The danger decreases, but not as much as we'd like to think. A great many computers on the Internet don't have their patches up to date; there are many examples of systems being broken into using vulnerabilities that should have been patched. I don't fault the sysadmins for this; there are just too many patches, and many of them are sloppily written and poorly tested. So while the danger decreases, it never gets back down to zero.

You can think of this as a graph of danger versus time, and the Window of Exposure as the area under the graph. The goal is to make this area as small as possible. In other words, we want there to be as little danger as possible over the life cycle of the software and the particular vulnerability. Proponents of bug secrecy and proponents of full disclosure simply have different ideas for achieving that.

History of Full Disclosure

During the early years of computers and networks, bug secrecy was the norm. When users and researchers found vulnerabilities in a software product, they would quietly alert the vendor. In theory, the vendor would then fix the vulnerability. After CERT was founded in 1988, it became a clearinghouse for vulnerabilities. People would send newly discovered vulnerabilities to CERT. CERT would then verify them, alert the vendors, and publish the details (and the fix) once the fix was available.

The problem with this system is that the vendors didn't have any motivation to fix vulnerabilities. CERT wouldn't publish until there was a fix, so there was no urgency. It was easier to keep the vulnerabilities secret. There were incidents of vendors threatening researchers if they made their findings public, and smear campaigns against researchers who announced the existence of vulnerabilities (even if they omitted details). And so many vulnerabilities remained unfixed for years.

The full disclosure movement was born out of frustration with this process. Once a vulnerability is published, public pressures give vendors a strong incentive to fix the problem quickly. For the most part, this has worked. Today, many researchers publish vulnerabilities they discover on mailing lists such as Bugtraq. The press writes about the vulnerabilities in the computer magazines. The vendors scramble to patch these vulnerabilities as soon as they are publicized, so they can write their own press releases about how quickly and thoroughly they fixed things. The full disclosure movement is improving Internet security.

At the same time, hackers use these mailing lists to learn about vulnerabilities and write exploits. Sometimes the researchers themselves write demonstration exploits. Sometimes others do. These exploits are used to break into vulnerable computers and networks, and greatly decrease Internet security. In his essay, Culp points to Code Red, Li0n, Sadmind, Ramen, and Nimda as examples of malicious code written after researchers demonstrated how particular vulnerabilities worked.

Those against the full-disclosure movement argue that publishing vulnerability details does more harm than good by arming the criminal hackers with tools they can use to break into systems. Security is much better served, they counter, by keeping the exact details of vulnerabilities secret.

Full-disclosure proponents counter that this assumes that the researcher who publicizes the vulnerability is always the first one to discover it, which simply isn't true. Sometimes vulnerabilities have been known by attackers (sometimes passed about quietly in the hacker underground) for months or years before the vendor ever found out. The sooner a vulnerability is publicized and fixed, the better it is for everyone, they say. And returning to bug secrecy would only bring back vendor denial and inaction.

That's the debate in a nutshell: Is the benefit of publicizing an attack worth the increased threat of the enemy learning about it? Should we reduce the Window of Exposure by trying to limit knowledge of the vulnerability, or by publishing the vulnerability to force vendors to fix it as quickly as possible?

What we've learned during the past eight or so years is that full disclosure helps much more than it hurts. Since full disclosure has become the norm, the computer industry has transformed itself from a group of companies that ignores security and belittles vulnerabilities into one that fixes vulnerabilities as quickly as possible. A few companies are even going further, and taking security seriously enough to attempt to build quality software from the beginning: to fix vulnerabilities before the product is

released. And far fewer problems are showing up first in the hacker underground, attacking people with absolutely no warning. It used to be that vulnerability information was only available to a select few: security researchers and hackers who were connected enough in their respective communities. Now it is available to everyone.

This democratization is important. If a known vulnerability exists and you don't know about it, then you're making security decisions with substandard data. Word will eventually get out -- the Window of Exposure will grow -- but you have no control, or knowledge, of when or how. All you can do is hope that the bad guys don't find out before the good guys fix the problem. Full disclosure means that everyone gets the information at the same time, and everyone can act on it.

And detailed information is required. If a researcher just publishes vague statements about the vulnerability, then the vendor can claim that it's not real. If the researcher publishes scientific details without example code, then the vendor can claim that it's just theoretical. The only way to make vendors sit up and take notice is to publish details: both in human- and computer-readable form. (Microsoft is guilty of both of these practices, using their PR machine to deny and belittle vulnerabilities until they are demonstrated with actual code.) And demonstration code is the only way to verify that a vendor's vulnerability patch actually patched the vulnerability.

This free information flow, of both description and proof-of-concept code, is also vital for security research. Research and development in computer security has blossomed in the past decade, and much of that can be attributed to the full-disclosure movement. The ability to publish research findings -- both good and bad -- leads to better security for everyone. Without publication, the security community can't learn from each other's mistakes. Everyone must operate with blinders on, making the same mistakes over and over. Full disclosure is essential if we are to continue to improve the security of our computers and networks.

Bug Secrecy Example

You can see the problems with bug secrecy in the digital-rights-management industry. The DMCA has enshrined the bug secrecy paradigm into law; in most cases it is illegal to publish vulnerabilities or automatic hacking tools against copy-protection schemes. Researchers are harassed, and pressured against distributing their work. Security vulnerabilities are kept secret. And the result is a plethora of insecure systems, their owners blustering behind the law hoping that no one finds out how bad they really are.

The result is that users can't make intelligent decisions on security. Here's one example: A few months ago, security researcher Niels Ferguson found a security flaw in Intel's HDCP Digital Video Encryption System, but withheld publication out of fear of being prosecuted under the DMCA. Intel's reaction was reminiscent of the pre-full-disclosure days: they dismissed the break as "theoretical" and maintained that the system was still secure. Imagine you're thinking about buying Intel's system. What do you do? You have no real information, so you have to trust either Ferguson or Intel.

Here's another: A few weeks ago, a release of the Linux kernel came without the customary detailed information about the OS's security. The developers cited fear of the DMCA as a reason why those details were withheld. Imagine you're evaluating operating systems: Do you feel more or less confident about the security the Linux kernel version 2.2, now that you have no details?

Full Disclosure and Responsibility

Culp has a point when he talks about responsibility. (Of course, Scott is avoiding "mea Culpa.") The goal here is to improve security, not to arm people who break into computers and networks. Automatic hacking tools with easy point-and-click interfaces, ready made for script kiddies, cause a lot of damage to organizations and their networks. There are such things as responsible and irresponsible disclosure. It's not always easy to tell the difference, but I have some guidelines.

First, I am opposed to attacks that primarily sow fear. Publishing vulnerabilities that there's no real evidence for is bad. Publishing vulnerabilities that are more smoke than fire is bad. Publishing vulnerabilities in critical systems that cannot be easily fixed and whose exploitation will cause serious harm (e.g., the air traffic control system) is bad.

Second, I believe in giving the vendor advance notice. CERT took this to an extreme, sometimes giving the vendor years to fix the problem. I'd like to see the researcher tell the vendor that he will publish the vulnerability in a few weeks, and then stick to that promise. Currently CERT gives vendors 45 days, but will disclose vulnerability information immediately for paid subscribers. Microsoft proposes a 30-day secrecy period. While this is a good idea in theory, creating a special insider group of people "in the know" has its own set of problems.

Third, I agree with Culp that it is irresponsible, and possibly criminal, to distribute easy-to-use exploits. Reverse engineering security systems, discovering vulnerabilities, writing research papers about them, and even writing demonstration code, benefits research; it makes us smarter at designing secure systems. Distributing exploits just make us more vulnerable. I'd like to get my hands on the people who write virus creation kits, for example. They've got a lot to answer for.

This is not clear-cut: there are tools that do both good and bad, and sometimes the difference is merely marketing. Dan Farmer was vilified for writing SATAN; today, vulnerability assessment tools are viable security administration products. Remote administration tools look a lot like Back Orifice (although less feature-rich). L0phtCrack is a hacker tool to break weak passwords as a prelude to an attack, but LC 3.0 is sold as a network administration tool to test for weak passwords. And the program that Dmitry Sklyarov was arrested for writing has legitimate uses. In fact, most tools have both good and bad uses, and when in doubt I believe it is better to get the information in the hands of people who need it, even if it means that the bad guys get it too.

One thing to pay attention to is the agenda of the researcher. Publishing a security vulnerability is often a publicity play; the researcher is looking to get his own name in the newspaper by successfully bagging his prey. The publicizer often has his own agenda: he's a security consultant, or an employee of a company that offers security products or services. I am a little tired of companies that publish vulnerabilities in order to push their own product or service. Although, of course, a non-altruistic motive does not mean that the information is bad.

I like the "be part of the solution, not part of the problem" metric. Researching security is part of the solution. Convincing vendors to fix problems is part of the solution. Sowing fear is part of the problem. Handing attack tools to clueless teenagers is part of the problem.

The Inevitability of Security Vulnerabilities

None of this would be an issue if software were engineered properly in the first place. A security vulnerability is a programming mistake: either an out-and-out mistake like a buffer overflow, which should have been caught and prevented, or an opening introduced by a lack of understanding the interactions in a complex piece of code. If there were no security vulnerabilities, there would be no problem. It's poor software quality that causes this mess in the first place.

While this is true -- software vendors uniformly produce shoddy software -- the sheer complexity of modern software and networks means that vulnerabilities, lots of vulnerabilities, are inevitable. They're in every major software package. Each time Microsoft releases an operating system it crows about how extensive the testing was and how secure it is, and every time it contains more security vulnerabilities than the previous operating system. I don't believe this trend will reverse itself anytime soon.

Vendors don't take security seriously because there is no market incentive for them to, and no adverse effects when they don't. I have long argued that software vendors should not be exempt from the product liability laws that govern the rest of commerce. When this happens, vendors will do more than pay lip service to security vulnerabilities: they will fix them as quickly as possible. But until then, full disclosure is the only way we have to motivate vendors to act responsibly.

Microsoft's motives in promoting bug secrecy are obvious: it's a whole lot easier to squelch security information than it is to fix problems, or design products securely in the first place. Microsoft's steady stream of public security vulnerabilities has led many people to question the security of their future products. And with analysts like Gartner advising people to abandon Microsoft IIS because of all its insecurities, giving customers less security information about their products would be good for business.

Bug secrecy is a viable solution only if software vendors are followers of W. Edwards Deming's quality management principles. The longer a bug remains unfixed, the bigger a problem it is. And because the number of systems on the Internet is constantly growing, the longer a security vulnerability remains unfixed, the larger the window of exposure. If companies believe this and then act accordingly, then there is a powerful argument for secrecy.

However, history shows this isn't the case. Read Scott Culp's essay; he did not say: "Hey guys, if you have a bug, send it to me and I'll make sure it gets fixed pronto." What he did was to rail against the publication of vulnerabilities, and ask researchers to keep details under their hats. Otherwise, he threatened, "vendors will have no choice but to find other ways to protect their customers," whatever that means. That's the attitude that makes full disclosure the only viable way to reduce the window of vulnerability.

In his essay, Culp compares the practice of publishing vulnerabilities to shouting "Fire" in a crowded movie theater. What he forgets is that there actually is a fire; the vulnerabilities exist regardless. Blaming the person who disclosed the vulnerability is like imprisoning the person who first saw the flames. Disclosure does not create security vulnerabilities; programmers create them, and they remain until other programmers find and remove them. Everyone makes mistakes; they are natural events in the sense that they inevitably happen. But that's no excuse for pretending that they are caused by forces out of our control, and mitigated when we get around to it.

Scott Culp's essay:
<http://www.microsoft.com/technet/columns/security/...>

Q&A with Culp:
<http://news.cnet.com/news/0-1014-201-7819204-0.html>

News articles on Culp:
<http://www.theregister.co.uk/content/55/22332.html>
<http://news.cnet.com/news/0-1003-200-7560391.html?...>
<http://cgi.zdnet.com/slink?153618:8469234>

Microsoft's push for secrecy:
<http://www.securityfocus.com/news/281>
<http://213.40.196.62/media/670.ppt>
<http://www.theregister.co.uk/content/4/22614.html>
<http://www.theregister.co.uk/content/4/22740.html>

My original essay on the Window of Exposure:
<http://www.schneier.com/crypto-gram-0009.html#1>

My earlier essays on full disclosure:
<http://www.schneier.com/...>
<http://www.schneier.com/...>
Note that the nCipher anecdote is untrue. Details are here:
<http://www.schneier.com/crypto-gram-0104.html#2>

Other commentary:
<http://www.securityfocus.com/news/270>
<http://web.ranum.com/usenix/ranum_5_temp.pdf>
<http://www.osopinion.com/perl/story/13871.html>
<http://www.synthesis.net/tech/fulldisclosure/>
<http://www.osopinion.com/perl/story/14401.html>
<http://www.net-security.org/text/articles/...>

Thanks to Tina Bird, Jon Callas, Scott Culp, Greg Guerin, Elias Levy, Jeff Moss, Eric Raymond, and Elizabeth Zwicky for reading and commenting on this essay.

Crypto-Gram Reprints

Why Digital Signatures are Not Signatures
<http://www.schneier.com/crypto-gram-0011.html#1>

Programming Satan's Computer: Why Computers are Insecure
<http://www.schneier.com/...>

Elliptic-Curve Public-Key Cryptography
<http://www.schneier.com/...>

The Future of Fraud: Three reasons why electronic commerce is different
<http://www.schneier.com/...>

Software Copy Protection: Why copy protection does not work:
<http://www.schneier.com/crypto-gram-9811.html#copy>

News

After all the posturing, Sen. Gregg is not going to introduce a bill mandating government access to cryptography:
<http://www.wired.com/news/conflict/0,2100,47635,00.html>

FBI is expanding its Internet wiretapping efforts:
<http://www.interactiveweek.com/article/...>