Global Information Assurance Certification Paper

Patch Me If You Can:
Patch Management's Vital Role in Defense-In-Depth

By

Bob Dehnhardt
SANS GIAC Security Essentials Certification (GSEC), v1.4b
October 15, 2003

Abstract

Keeping systems up to date with patches and service packs is an effective way of preventing virus and worm infections. But experience shows that patches are frequently ignored, or applied inconsistently. Worms and viruses spread rapidly, even when the patches have been available for 6 months or more. Many experts are blaming the software manufacturers for buggy software, but the issue of patch management remains to be addressed. This paper will look at the causes of this problem, and offer some solutions to help enable effective patch management and deployment in a corporate environment.

From Melissa, With Love…

In January 1999, Microsoft published a bulletin describing a vulnerability in their Word 97 program that could allow the running of unauthorized macros. This security bulletin also contained a link to a patch that would correct the problem and close the vulnerability. In March 1999, the Computer Emergency Response Team/Coordintaion Center (CERT/CC) released an advisory describing a macro virus that exploited this vulnerability. The virus was dubbed Melissa, due to an entry it made in the Windows registry, and it spread by running a macro that would automatically email itself to addresses in the user's address book or Contacts list.

Although more than two months had passed between the release of patch and exploit, Melissa spread widely and rapidly, getting more mainstream press coverage than any previous computer virus. Despite this notice, the virus continued to spread for several months, finding more and more unpatched systems and virgin territory.

Fast forward to July 2003. Microsoft published a security bulletin describing a vulnerability in the RPC interface that could allow an attacker to run code with Local System privileges. The bulletin also contained links to a patch that would correct the problem and close the vulnerability, as well as workarounds that would help protect vulnerable systems until the patch could be applied. Later that month, CERT/CC released an advisory warning of increased traffic scanning for this vulnerability, and advising people to patch their systems immediately. Less than two weeks later, another CERT/CC advisory was released, announcing the Blaster worm (sometimes called the LoveSAN worm), which used this vulnerability to spread. Fast on this worm's heels came a variant dubbed Nachi, a "noisier" worm that used ICMP ping packets to find hosts, and would exploit an older vulnerability if the RPC hole were patched.

More than four years had passed between Melissa and Nachi. Those intervening years saw the release of worms and viruses with names like Code Red, Nimda,

sadmin/IIS and SQL_Slammer. In each of these cases, the vulnerability had been discovered and a patch made available in advance of the worm's release, sometimes by several months. In each of these cases, the malicious code spread quickly, finding thousands of unpatched systems available. Two years after their release, the Code Red and Nimda worms are still active.

The evidence implies that the situation has not improved substantially in four years. Patches are not getting installed quickly enough, if at all. Vulnerabilities exist far longer than they need to, giving virus and worm writers greater opportunity to exploit them. Nachi's use of a months-old exploit as a fallback infection method shows that virus writers expect vulnerabilities to go unpatched. Companies spend thousands of dollars or more on cleanup and disinfection, and lose millions in lost revenue and productivity, when prevention can cost relative pennies. Why does this situation continue?

Patch Management's Role

Before we consider this question, let's first look at patch management's place in a Defense-in-Depth strategy.

In his presentation on adopting a balanced approach to security, IBM Security Specialist Joseph F. Martin describes Defense-in-Depth as a stack of Swiss cheese slices. Each slice has holes of varying size and shape, and by carefully selecting the slices, the holes in one layer are covered by cheese in another. No security device, application, or implementation is or can be perfect; they all have holes. But by carefully selecting your security devices and applications, configuring them properly, implementing them in accordance with a well-designed plan, and updating them as necessary to adapt to the ever-changing environment, you can minimize these holes and maximize your security "cheese."

Martin also describes how Defense-in-Depth can be further separated into the areas of Prevention, Detection and Response, with each area having its own layers. Prevention covers the items most people usually think of when discussing network and information security: firewalls, anti-virus, vulnerability scanning, backups, passwords, and user rights management, to name a few. These are the items that have the highest profile, and generally receive the most funding. Detection provides eyes into your network and system space, allowing you to see what's happening and respond appropriately. This area include network- and host-based Intrusion Detection Systems, as well as log analysis, and web, email and instant messenger traffic monitors. And Response covers how a team deals with the security incidents that do happen. This area will generally include forensics tools and data collected from the Detection area, plus policies and procedures designed to carry the team through their investigation.

System patches can be thought of as one of the most fundamental layers of preventative defense. Unlike firewalls that block traffic based on defined parameters, or anti-virus software that scans for known code signatures, patches actually remove the problem by correcting the flaws in the application of operating system software. A Windows system patched with MS003-026 and MS03-007 would be immune to the MSBlaster and Nachi worms. Linux systems running OpenSSL version 0.9.7a are immune to certain ASN.1 encodings that could crash earlier versions. Patches can be a powerful defense, but they only work when they are installed.

Why Don't They Do It Right The First Time?

Why do we have to patch, anyway? Can't they just get it right the first time? Why am I paying good money for software that has so many bugs in it? They should just wait until it's all fixed, until everything is working properly, and then release it. I'm not a beta tester, I'm an end user; I deserve a product that works.

Anyone who has spent much time in desktop support has heard it, some of us have even thought it. If a car is found severely defective, "lemon laws" entitle us to replacement. If a toaster doesn't work when we get it home, we return it to the store and get a new one. After the recent spate of patches and worms, articles appeared blasting Microsoft, and asking the question: Why can't we get the same guarantee of quality in software?

These articles are quite correct in saying that software vendors, and especially Microsoft, could do a better job in removing known bugs and vulnerabilities before shipping their software. However, they go too far when they suggest that software should be perfect when shipped, the manufacturer bears all responsibility for patching, or that patch management is a waste of time.

The truth is, patches have been around almost as long as the programs themselves. The Free Software Foundation's web site contains an undated, anonymous poem that tells the story of a programmer determined to completely debug his program before releasing it to the world. He is eventually found at his console, surrounded by memory dumps and punch cards, dead from starvation.

> And the last bug in sight,
> An ant passing by,
> Saluted his tombstone,
> And whispered, "Nice try."

The poem was meant to be humorous, but it also points out the futility of trying to completely debug software. There will always be bugs that slip past the most diligent, comprehensive testing procedures. And although Microsoft has borne the brunt of criticism recently, it is important to remember that virtually all software contains bugs, whether internally developed, Open Source, or closed-

source commercial products. In his essay “The Cathedral And The Bazaar”, Eric S. Raymond notes of Open Source, “Given enough eyeballs, all bugs are shallow.” He uses this statement to show how the Open Source methodology can enable faster and more complete debugging. But even the thousands of eyeballs scrutinizing the Linux kernel can’t prevent bugs from making it into released versions. And as software grows in complexity, it stands to reason that bugs will grow in number and subtlety. Bugs happen, and patches are needed to correct them.

In general, bugs fall into two groups: Functional and Security. Functional bugs are found in features that simply don’t work as designed (for the purposes of this document, I’m excluding features that don’t work the way the user thinks they should). Patches for these bugs contain code that restore normal functionality to the features, and are what we normally think of as “bug fixes.”

Security bugs are in many ways subtler, and far more serious, than Functional bugs. They can exist in software that is perfectly functional, but can grant users rights and access levels that they shouldn’t have. These bugs can be caused by miswritten or poorly written code, or by design, implementation or configuration flaws that didn’t appear in testing. Patches may simply include code corrections, or a complete rewrite of the application. The extent of the patch will depend on how fundamental the flaw is.

Why Johnny Won’t Patch

So if buggy software is unavoidable, and these vulnerabilities can leave your systems wide open, then everyone must be diligently applying patches in a timely manner, right? Well, no. The fact is, most systems will go for months, or even years, before even the most fundamental of patches are applied.

This situation is not new. In The Cuckoo’s Egg, Cliff Stoll documents his experiences tracking “black hat” hackers working for the KGB in 1988. The hackers broke into Stoll’s system in the Lawrence Berkeley Labs by exploiting a vulnerability in the GNU-Emacs editor. A patch for this vulnerability was trivial, but hadn’t been applied in Berkeley. Using this as a starting point, the hacker was able to gain access to systems at MIT, SRI International, Air Force Space Command, and a number of other military and defense contractor sites. A number of these sites were running the same unpatched version of Emacs as Stoll.

Patches are necessary. They’re inevitable. They provide valuable protection. So, why don’t they get applied?

Notification is one of the oft-mentioned reasons. Users, especially those in home environments, don’t always get notified that patches are available, or that their system needs them. This problem was more prevalent in the past, but software

vendors have recently taken steps to address this. Beginning with Windows 2000, Microsoft has included a Windows Update function that will notify the user when patches that apply to that user's system are available. Most Linux distributions, like Red Hat, have similar mechanisms. The operating systems ship with the function turned on by default, although the user can easily disable it.

Another reason given is that the patches are too difficult and labor-intensive to install. This was certainly the case with MS02-039, which patched the Microsoft SQL Server vulnerability exploited by the SQL_Slammer worm. The initial release of this patch could overwrite files from a different security patch, effectively restoring a vulnerability to your system. The patch also came with no automated installer, and had to be applied manually, with the user stopping services and copying files as instructed by a README file. But this reason is given even in cases where automated installation is available – IT staff must go from machine to machine, running the automated process, and making sure it completed successfully. Normally, this kind of work must be done outside of normal business hours, and can quickly add up to significant overtime costs.

"Seems like every time I install a system patch, something else goes wrong with my system." This quote from Frank Beier, president of Dynamic Webs, echoes a common complaint about patches in general, and Microsoft patches in particular: the patches seem to break as much as they fix. Most of these problems seem to revolve around locally developed applications, but there have been cases where a Microsoft operating system patch has caused commercial software (including Microsoft's) to fail.

Effective Patch Management

Writing in Computerworld, Alex Bakman, CEO of Ecora Software, offers five tips for effective patch management that address these issues. They are:

1. Automate
2. Plan your approach
3. Test patches
4. Know the configurations
5. Maintain patch levels

I would add two more tips to this list:

6. Document your deployments
7. Communicate, communicate, communicate!

Let's examine each of these tips in more detail.

Automate – The days of "sneakernet", running from PC to PC with a floppy disk, are well behind us. Gigabit Ethernet to the desktop is becoming more feasible

every day, and IT managers need to put this resource to use. Automation of patch analysis and deployment can significantly reduce the time and effort involved in deploying patches, as well as simplify their application.

Two well-regarded patch automation products for the Microsoft Windows environment are LANGuard Network Security Scanner from GFI (www.gfi.com) and HFNetChkPro from Shavlik Technologies (www.shavlik.com). Both of these products will scan systems on your network for vulnerabilities and missing patches or service packs, push the patches out to designated systems, and activate the installation. These tools have the latest patch information from Microsoft, so your scans are always current. A few mouse clicks from a single workstation can replace many hours of overtime and multiple system touches. The ability to scan for patches allows the security team to quickly assess their vulnerability to a newly announced exploit, especially in cases where the patch has been available months in advance.

In my opinion, the HFNetChkPro product scales to larger environments better than LANGuard, although both will support any number of clients. HFNetChkPro includes a “test group” function, which allows you to identify systems for evaluating new patches before deploying them to the general populace, as well as support for patching all Microsoft product lines. It is also more expensive, working on a by-seat licensing structure with no “unlimited” option. Either one, however, will quickly pay for itself in short order, when labor savings from patch deployments are taken into consideration.

Both companies offer free versions for evaluation and use in small networks or non-corporate environments. And there are many other products out there that offer other features and functions. Any of these will help reduce the time, labor and difficulty in deploying patches to your user base.

Plan you approach – One mistake that is commonly made is to view the environment as one large group, and assume that everything needs to get patched all at once. My father-in-law likes to say “The best way to eat an elephant is one bite at a time.” Dietary considerations aside, this makes good sense. Reducing a large task into smaller bites makes the overall task more manageable.

Divide your systems into sensible groups – location, function, department, business processes, etc. Use whatever divisions make sense for your organization. Ideally, you should use multiple groupings – a system could be a member of the “servers” group, as well as the “web farm”, “Oakland office” and “customer facing” groups.

When a patch is released, you can then prioritize your deployment based on the risks associated with each group. Patch the highest risk group first, and on down the line.

Test patches – Designate a number of systems to act as a test base, and apply your patches here first before rolling them out to the general populace. Patches can and often do change APIs, registry entries, and system functionality, which can lead to problems with other applications. Your test group can consist solely of "lab" equipment, but ideally, it should include some actual production systems, especially workstations. It should be representative of your total computing environment, and include all mission-critical processes and applications.

Establish a clear testing procedure that covers as much functionality as possible. Also establish a timeline for testing, and acceptance guidelines. These should define the test period, and provide a means for determining if an incompatibility with the new patch is a showstopper. By establishing these guidelines in advance, your testing period can be smooth, thorough and brief.

Know the configurations – Beyond keeping your lab and production equipment in sync, you should also keep close track of what applications are installed in your environment, and where. This can help you target patch deployments, further reducing the time investment into the process.

Maintain patch levels – Playing catch-up is a painful process, even with automation software. But once you are caught up, it is far easier to maintain that level than to backslide and catch up again. Staying on top of patch levels by regularly scheduled scanning and deployment will save you time and worry in the long run.

Document your deployments – Keep a log of your deployment activities, as detailed as you can. Many automation packages do this automatically; if possible, export these automatic logs to a location that is accessible by IT staff. This information can help in diagnosing system problems (symptom "A" was first observed 2 hours after patch deployment), as well as identifying possible user mischief or system compromise (patch was successfully deployed on day "B", but was missing when system was scanned two days later).

Communicate, communicate, communicate! – At the very least, you should inform IT staff and upper management when vulnerabilities are identified and patches are deployed. Depending on your organization's policies on email announcements, broadcast emails regarding critical vulnerabilities and patches should be made to all staff. If nothing else, doing so raises awareness of security issues within your organization, and provides an opportunity for other security-based discussions. You should never waste an opportunity to get people thinking (in a positive way) about security.

The Politics of Patching

Another question that must be addressed is that of ownership: Who Owns Patching? If your security and systems administration staffs are one and the same, this is an easy one to answer. But if your organization has separate staffing for security and administration, this question can become a political powder keg.

Consider the following situation. A mid sized company has a mixed computing environment. Desktops are mainly running Windows 2000, as are the file and mail servers, domain controllers, and some other special-purpose servers. Back office databases and application engines run on minicomputers from two different vendors, running the vendor's flavor of Unix. Several Linux boxes are scattered about, as well as a few Macs running OS X. In all, a thoroughly mixed environment.

On the staffing side, one group is providing systems administration and desktop support, while another group is handling network and information security. Both groups have root-level access to all systems, and both have administration experience. So who owns the patch that was just released?

Well, both would have a valid claim. Security should own the process, since these patches address vulnerabilities that need to be guarded against. In order to properly assess any threat to the environment, security staff must know what vulnerabilities exist, and patches definitely play into that equation. At the same time, patches can (and often do) change the functionality of an application or operating system. SysAdmins are in the best position to judge the impact of a patch on a system; they know what's installed, and the dependencies involved. It is my opinion that patch management needs to be owned by both groups, with careful coordination and a clear breakdown of duties and responsibilities along the lines described below, communicated to all members of both teams.

- Security is responsible for assessing the risk exposure to the vulnerabilities the patches address, and suggesting a deployment priority and schedule. The assessment would take into account such factors as ease and likelihood of exploit, installed base of the affected software, impact of those systems to mission-critical processes, and exposure of those systems to attack. The final assessment could be a one-line summary, something like "Minimal risk - schedule deployment as convenient", "Low risk - recommend deployment during next off-cycle weekend", "Moderate risk - recommend deployment within 3-4 days", "High risk - recommend deployment within 48 hours", "Extreme risk - recommend deployment within 24 hours", or "Critical risk - recommend no testing, interrupt production, deploy immediately", with additional detail and explanation as needed.
- SysAdmin is responsible for testing, coordination of fixes and workarounds to affected apps, and actual scheduling and deployment of the patch. This puts

SysAdmin in the driver's seat - if tests show that the patch would significantly impact production (rendering a production database unavailable, or crashing all customer-facing web servers), SysAdmin notifies Security of the problem and scheduling delay. This way, Security is informed of the system and network status, and can respond appropriately to further information about this particular vulnerability. Security can also look for alternate ways to defend against any exploits that may be coming.

- SysAdmin notifies Security upon successful deployment of the patch. Again, this allows Security to assess and respond to exploits, worms, and intrusion attempts with a full understanding of their environment.
- Security will maintain tools for ad hoc checking for patch levels on various systems. This gives Security independent confirmation on patch status, should any questions or concerns arise.

Sharing ownership of patch management allows system maintenance to remain with those most familiar with the computing environment, but it still provides for risk assessment and prioritization. As a side benefit, it raises awareness of security issues with the SysAdmin group, which is always a good thing. Getting the system maintainers to think about security in one area will often spill into other areas, and can lead to a stronger overall security focus.

No Silver Bullet

Patches are essential to keeping your systems and devices secure and functional, and effective patch management provides the mechanism for this vital layer of defense. Software tools can aid in this management, or “sneaker-net” can be relied upon, as long as the patches are getting applied.

But it is important to remember that while patches often constitute the last line of defense against most exploits, they are not the fabled “Silver Bullet” of security. A properly patched system is stronger and more resilient than an unpatched one, but it is not invulnerable.

On October 1, 2003, a new exploit for Microsoft’s Internet Explorer was announced. This exploit was similar to one patched the previous August, but that patch didn’t apply to the new exploit. Microsoft and security experts were unaware of this new vulnerability, and for almost 60 hours, there was virtually no defense. Effective patch management cannot prevent zero-day exploits from getting on your system, but it can help you to quickly respond once a patch becomes available.

The recent flurry of articles has raised consciousness regarding patches and their place in security, but has also muddied the waters by trying to place blame for the problem with the software manufacturers. Rather than trying to assign blame or wait for someone else to solve the problem, I think the better response is to accept responsibility for the maintenance of the systems under your control,

and use the resources and tools available to you to handle the problem yourself. Patches will always be with us. “Patch management” is not an oxymoron, even with the number of patches being released today. Instead, patch management is a basic part of system maintenance, and a vital layer of a sound defense strategy.

George Santayana said, “Those who do not learn from history are doomed to repeat it.” We have seen this happen, from well before when Melissa sent her calling card, through when SQL_Slammer set propagation records, and as recently as MSBlaster and Nachi tag-teaming through the network. And it continues: On September 10, 2003, Microsoft published a bulletin describing a vulnerability in the RPCSS service of the Windows NT, 2000, XP, and 2003 Server editions of their operating system. This vulnerability was similar to the one exploited by the MSBlaster and Nachi worms, which raised concerns that an exploit would be quick in coming. Microsoft released a patch for this vulnerability immediately. One month later, on October 10, 2003, reports appeared that exploit code for this vulnerability was circulating, and a new worm that might operate in spite of the patch could appear at any time.

A number of IT and Security managers were blissfully unaware of this, and had no idea as to what was happening. Others began calling their family and friends to cancel plans, gearing up for yet another all-nighter. And still others took a few minutes out of their day to calmly confirm that their patch levels were appropriate, and their other security layers were in place and functioning properly.

Which group will you be in?

Works Cited

Anonymous. “The Last Bug.” Free Software Foundation. http://www.gnu.org/fun/jokes/last.bug.html

Bakman, Alex. “Five tips for effective patch management.” *Computerworld*. 14 July 2003. http://www.computerworld.com/securitytopics/security/story/0,10801,82458,00.html

Barney, Doug. “Microsoft Reacts to Security Stings with Tools, Promises, and Mea Culpas.” *ENT News*. 9 October 2003. http://entmag.com/news/article.asp?EditorialsID=5983

Briney, Andrew. “The folly of patch and proceed.” SearchSecurity.com Newsletter. 6 October 2003.

Common Vulnerabilities and Exposures (CVE) list. *CAN-2003-0543: Integer overflow in OpenSSL 0.9.6 and 0.9.7.* 14 July 2003. http://cve.mitre.org/cgi-bin/cvename.cgi?name=can-2003-0543

---. *CAN-2003-0544: ASN.1 parsing vulnerability in OpenSSL 0.9.6 and 0.9.7*. 14 July 2003. http://cve.mitre.org/cgi-bin/cvename.cgi?name=can-2003-0544

---. *CAN-2003-0545: Double-free vulnerability in OpenSSL 0.9.7*. 14 July 2003. http://cve.mitre.org/cgi-bin/cvename.cgi?name=can-2003-0545

Computer Emergency Response Team/Coordination Center (CERT/CC). *CA1999-04: Melissa Macro Virus.* Pittsburgh: 27 March 1999. http://www.cert.org/advisories/CA-1999-04.html

---. *CA-2003-19: Exploitation of Vulnerabilities in Microsoft RPC Interface*. Pittsburgh: 31 July 2003. http://www.cert.org/advisories/CA-2003-19.html

---. *CA-2003-20: W32/Blaster worm.* Pittsburgh: 11 August 2003. http://www.cert.org/advisories/CA-2003-20.html

Cooper, Charles. “Software security running out of excuses.” *ZDNet UK*. 18 August 2003. http://comment.zdnet.co.uk/charlescooper/0,39020664,39115766,00.htm

Fisher, Dennis, and Chris Gonsalves. “Internet Recovering From Slammer.” *EWeek.* 25 January 2003. http://www.eweek.com/article2/0,4149,845166,00.asp

Lemos, Robert. “Microsoft fails Slammer’s security test.” *C|Net News.com*. 27 January 2003. http://news.com.com/2100-1001-982305.html

Martin, Joseph F. “Protecting Your Critical Assets: The Value of a Balanced Approach.” Presentation. Las Vegas/Reno InfraGard meeting. 18 September 2003.

McAlearney, Shawna. “Zero-day IE exploit just the beginning.” *Information Security Magazine*. 1 October 2003. http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci930187,00.html

---. “Update: Exploit code targets recent RPC flaws.” *Information Security Magazine*. 10 October 2003. http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci931798,00.html

Microsoft Corporation. *MS99-002: Word 97 Template Vulnerability.* Seattle: 21 January 1999. http://www.microsoft.com/technet/security/bulletin/MS99-002.asp

---. *MS03-007: Unchecked Buffer In Windows Component Could Cause Server Compromise.* Seattle: 17 March 2003. http://www.microsoft.com/technet/security/bulletin/MS03-007.asp

---. *MS03-026: Buffer Overrun In RPC Interface Could Allow Code Execution*. Seattle: 16 July 2003. http://www.microsoft.com/technet/security/bulletin/MS03-026.asp

---. *MS03-039: Buffer Overrun In RPCSS Service Could Allow Code Execution*. Seattle: 10 September 2003. http://www.microsoft.com/technet/security/bulletin/MS03-039.asp

Raymond, Eric S. “Release Early, Release Often.” *The Cathedral And The Bazaar*. 1997. http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html

Stoll, Cliff. The Cuckoo’s Egg: Tracking a Spy Through the Maze of Computer Espionage. New York: Pocket Books, 1989.