Let’s Slam SQL: The Slammer Worm and Lessons Learned

Brian Greif
GSEC v1.4b
Option 1

Abstract

One of the most frightening events dealt with by today’s network engineers and administrators it the realization that there is a worm or virus on their network. These infestations are some of the most frustrating, time consuming, and expensive areas of technical support today.

In this paper, the basic history of the Internet worm will be covered, as well as one specific worm attack in detail. The methods by which some of these worms work will be also be explored. The use of certain tools that can be used in order to safeguard networked systems will also be discussed. Finally, a brief overview will be given, regarding some best-business practices in order to help administrators deal with worm or virus attacks in a more efficient manner.

Discussion

Since the inception of this “network of networks”, the Internet has been an “interesting” place to be. Although never truly thought of as a safe place, at one time it was an open group of systems where, for the most part, everyone trusted everyone else. In a system inhabited mostly by academics and geeks, the majority of its users would have never thought to intentionally release any kind of harmful code or program.

That all changed on November 2, 1988[1], when Robert Morris Jr. released his worm into the wild. Mr. Morris was a graduate student in Computer Science studying at Cornell University. He wrote a self-replicating, self-propagating program (the worm), and sent it out to the Internet. Curiously, he chose to release it from MIT rather than Cornell. The program exploited holes in the sendmail and fingerd programs, as well as in rsh/rexec, and also exploited weak passwords. The problem was that the program had a bug that made it consume the infected computer system’s resources until it was unable to function normally. This was more destructive and caused more damage than Mr. Morris had anticipated. Eventually, Morris’ worm caused many computers connected to the Internet to crash or hang.

The effected computers were attached to university, military and medical research networks, among others, and when Mr. Morris noticed how fast his program was spreading, he called upon a friend from Harvard to help him find a way to stop the worm from causing more damage. They eventually sent out a message to the Internet instructing programmers on how to kill the worm and

[1] Kehoe

prevent re-infection, but unfortunately, the message did not get through until it was too late. Many facilities had already taken their machines off-line, and in addition, due to the traffic that the worm had created on the Internet, other facilities had trouble receiving the message sent to help. By the time it was all over, the Morris worm had cost an estimated 6,000 victims between $200.00 and $53,000.00 in damages[2]. Mr. Morris was charged with and later convicted (Morris being the first conviction) of violating the Computer Fraud and Abuse Act (Title 18). This act criminalizes the act of misusing and/or causing damage to a computer or system either physically or by the use of harmful code.

Since then, worms and viruses have been some of the biggest "spotlight hogs" and buzz generators in regard to the Internet and network security. One of the core differences between worms and viruses is that a virus generally infects files on one local machine and requires human interaction to go from one machine to another; a worm does not. For example, if one were to receive an infected program as an email attachment, the payload would not be released until the user ran the infected executable. A worm, on the other hand, is designed to be self replicating[3] and autonomous, and is usually coded in order to exploit holes in commonly run daemons or services. A worm may also exploit poorly chosen passwords.

Once a worm has exploited a programmatic hole, it proceeds to scan the network for other machines offering the exploitable service. While a virus sometimes has a destructive "payload" that may destroy data, a worm usually causes havoc based on the amount of traffic it generates due to its scanning capabilities.

These destructive programs can sometimes move extremely fast. One such worm that reared its ugly head in recent history was the Sapphire, or Slammer worm.

A Little Bit of Background on Slammer

The Slammer worm was released on the Internet (apparently from the Asian region) and started to infect computer systems at approximately 05:30 UTC on Saturday, January 25th 2003. As it propagated itself across the Internet, it began growing at an incredible rate, doubling in size every 8.5 seconds for the first minute. Within 10 minutes it had already infected 90% of the vulnerable systems on the Internet. These vulnerable systems consisted of machines running Microsoft SQL Server or MSDE 2000 (Microsoft SQL Desktop Engine). By the time the worm began to slow down, it had infected over 74,000 (probably considerably more) distinctly addressed computers and systems. Slammer has been classified as the fastest spreading worm in the history of the Internet. It has also been spoken of as a true "Warhol" worm[4]. The name refers to Andy

[2] Kehoe
[3] Simenin
[4] Weaver

Warhol’s quote: “[i]n the future, everybody will have 15 minutes of fame”, and infers that the worm had made a tremendously significant impact on the Internet within 15 minutes. The fact that the worm did not need human intervention in order to fulfill its programmatic duty is one reason for this. Another reason for the speed of the worm’s movement is that it was a small worm containing a compact, efficient scanner. The worm’s total size amounted to only 376 bytes, and with the UDP headers the total payload was 404 bytes.

Fortunately, Slammer did not have a malicious payload. It did, however, cause serious damage by making use of as much bandwidth as possible in order to scan other systems. It also had the unfortunate effect of taking many database servers out of commission.

The Inner Workings of Slammer

The Slammer worm, as previously stated, worked its way around the Internet very quickly. Slammer was a well-crafted worm that took advantage of the fact that the SQL exploit could be fit into 1 UDP packet with the size of 404 bytes. This is a relatively small packet when compared to other previously seen scanning worms. For example, Code Red’s payload was a full 4kb, and the payload associated with the Nimda worm was a full 60kb.[5] Noting this payload size, it is obvious that the Slammer worm was built for speed. Slammer was also faster than previous worms due to the methodology in which network probes were initiated. Code Red spread itself by opening many threads, with each thread opening its own connection sequence in order to scan more addresses. The worm then had to take the time to wait for the TCP SYN/ACK to happen if there was a response from the scanned IP address. Because of this, Code Red could be considered latency limited.

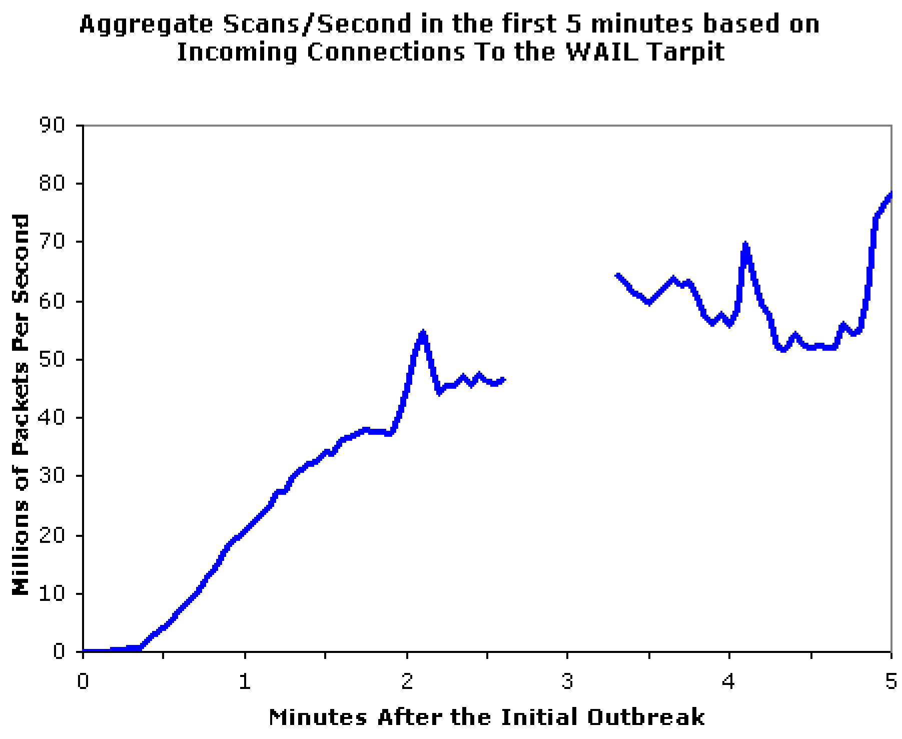
By contrast, the Slammer worm is considered bandwidth limited. Slammer fit its entire payload into one small packet that was sent to UDP port 1434 and did not have to wait for a response. This is due to the fact that UDP is a connectionless protocol and does not have to wait for a handshake sequence to be completed. Due to the fact that Slammer did not have to wait for a response, it could send out a large number of packets extremely quickly. The Cooperative Association for Internet Data Analysis (CAIDA) reported that the largest single probe rate observed during Slammer’s most active period on the Internet was 26,000 scans per second, with an average Internet-wide probe rate of approximately 4,000 scans/second per worm during its initial phase of growth.[6] Although it sounds incredible, Slammer’s scanning rate peaked in under three minutes with an aggregate count of an amazing 55 million scans per second. Ironically, the fact that the worm was so fast, and sent out packets so efficiently, eventually began to work against itself as the bandwidth available became smaller and smaller.

[5] Moore
[6] Moore

Slammer propagated itself using a technique called random scanning. This means that the worm picked random IP addresses to scan. Given enough time on the Internet, it is assumed that it would eventually find all hosts that were susceptible to the exploit. This method of scanning however, eventually helped to slow down the worm's rate of infection as it performed repeat scans of previously scanned and infected hosts, as well as continuing to scan machines and nodes that were unexploitable to begin with. The scanning portion of Slammer also had some other interesting qualities. In order for a worm to use the technique of random scanning, it must use a randon number generator. The Psuedo Random Number Generator (PRNG) that was written into Slammer was flawed, and "[would] or [would] not include entire /16 blocks of IP addresses in a cycle."[7] This means that significant numbers of Internet IP addresses were not scanned. Although this represents a problem in the PRNG and makes it harder for the global Internet community to track all scanned and exploited addresses, this flaw was probably helpful in reducing the total amount of damage incurred by Slammer.

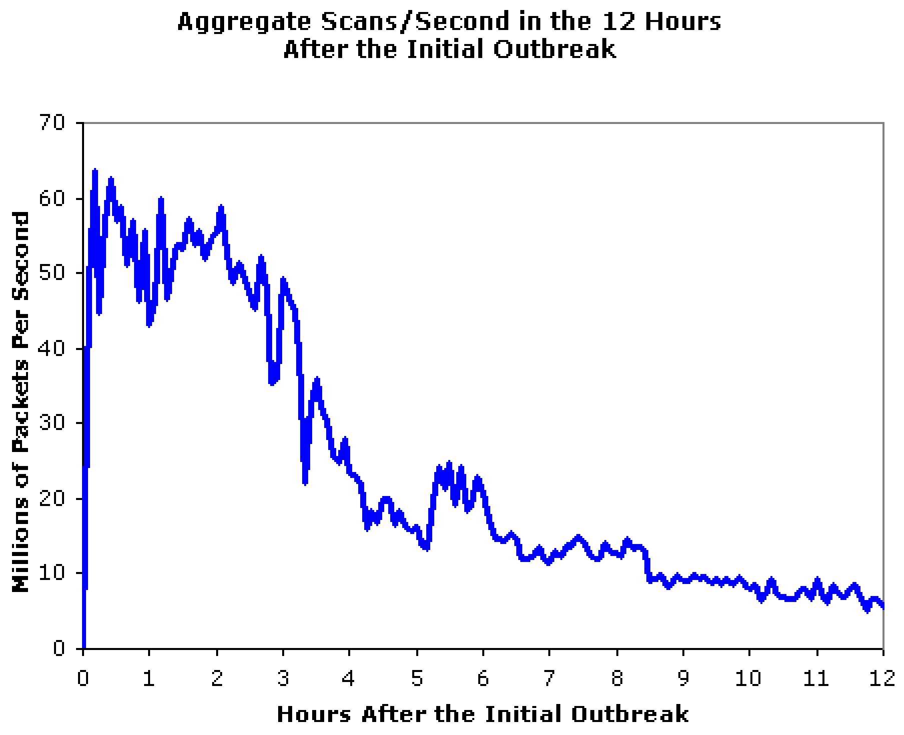
Timeline of Attack and Proliferation

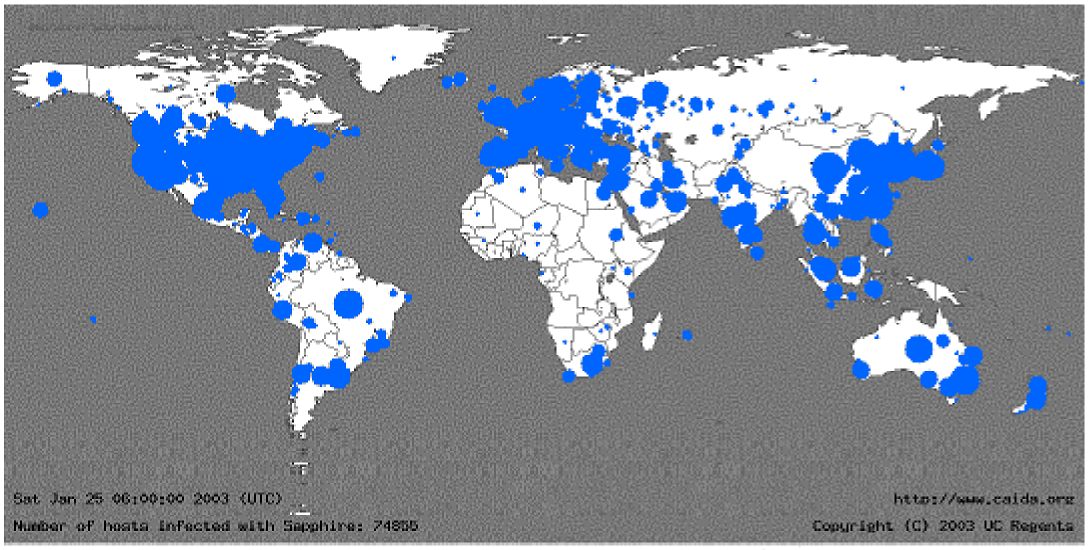
The following graph illustrates the rate of speed with which Slammer took off once it was released on the Internet. There was a transient failure in the University of Washington Advanced Internet Lab (WAIL)'s data collection at the 2 minute and 40 second mark that is represented by white space.



[7] Moore

This next chart shows the response to Slammer and the rate at which the worm's scanning and proliferation slowed down over the 12 hours immediately following its release. The chart helps to illustrate the fact that while Slammer was extremely fast and prolific, network administrators were able to begin taking care of the problem quite quickly. Fortunately, many administrators were able to quickly stop the scanning from continuing to enter their sites by filtering (blocking) all traffic from UDP port 1434. This would have been harder to do if Slammer had used a port that is used for critical, legitimate Internet traffic such as TCP port 80 (WWW) or UDP port 53 (DNS).




Sat Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Sapphire: 74855
http://www.caida.org
Copyright (C) 2003 UC Regents

The chart above graphically represents the geographic spread of Slammer in the thirty minutes after it was released. All three above charts are taken from http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html, Moore *et. al* (January 2003)

Although Slammer did not crash HTTP servers specifically, it created so much traffic on so many different networks that when trying to access web servers, they themselves seemed unreachable. If Slammer had been released on a Monday instead of a Saturday, the costs of fixing the damage done would have been considerably higher, if only because of the fact that so many companies use the Internet to conduct their daily business. Unfortunately, many businesses were impacted even though the attack happened on a weekend. Some of these businesses were critical services such as health-information network databases, as well as ATM and other financial databases.

Mitigating the Risk of the Slammer Worm on Your LAN

The best way to keep a network secure and safe from the damage caused by a worm like Slammer is of course not to let it in at all. There are many ways to do this, and include things like filtering TCP port 1433 and UDP port 1434 at the perimeter and/or desktop firewall (as stated earlier), or running your database application with as few permissions as possible[8]. Applying system and security patches when feasible also makes good sense.

One reaction to hearing about all of these problems is to say "just don't run SQL Server." An interesting observation was made however, when looking at some of the companies that were impacted by the Slammer worm. Namely, how was so much damage done to various networks and systems that were not running Microsoft SQL Server? The one word answer is MSDE. The Microsoft SQL

[8] Johnson

2000 Desktop Engine is installed with a large assortment of software packages by default[9]. Some of these applications are financial management packages, network management and discovery tools and other programs that seem quite innocuous. If an administrator had installed any of these applications and his/her systems were not patched, the chances of compromise were quite high. Basically, administrators need to be aware of any database-connected applications.

One surprising fact regarding the Slammer worm and the programs that it exploited is that Microsoft had released a patch to fix this problem on October 16th, 2002[10]. It is interesting to note that a patch that was made public more than six months before Slammer became a reality was not more widely implemented. Even Microsoft itself suffered from the Slammer worm exploiting and infecting unpatched systems on its network.

Once Slammer has hit a LAN or computer system, the best course of action would be to disconnect the infected computers, shut down the SQL processes, and remove the worm from memory. This could be accomplished in a number of ways. For users of Network Associates AVERT product, a program named Stinger will clean the worm and shut down SQL processes on an infected server. Trend Micro users can use the System Cleaner product from the Trend Micro website[11]. One important thing to consider is that after the worm has been cleaned from an infected systems memory, the Microsoft SQL patch must be applied in order to keep the system free from re-infection.

Lessons Learned

It seems disturbing that all this damage and lost revenue occurred due to a small program that exploited a flaw that had a previously released patch. As stated earlier, even Microsoft succumbed to the effects of the Slammer worm. So what are the already overworked LAN/WAN administrators, network security professionals, and helpdesk personnel supposed to do regarding security patches and updates, whether worm-based or otherwise? Part of the answer to that question lies in proper preparation and planning. Although it is a clichéd expression that "those who fail to plan, plan to fail", this statement really does ring true in the network security arena.

Planning for Disaster

The first part of having a good, solid plan for mitigating the risk associated with security patches or worms and viruses in general is knowing what you have running on your systems and network. If a worm that exploits a certain vulnerability is released on the Internet, is there a process in place in order to

[9] sqlsecurity.com
[10] microsoft.com
[11] zdnet.com

figure out whether or not the systems and networks under your care will be affected? One needs to know what runs on his or her systems. Each system needs to be fully documented as to what its main functions are. It is sometimes an even bigger help if major, similar services can be consolidated by machine. For example, if possible, all web services can be served from one or two machines. This applies to Oracle or SQL databases as well. This works on the premise that, for example, if one knows that all of the Apache based web services are being offered from one or two machines, it is fairly easy to know which machines you need to patch when the Apache group releases a bug fix. Although this seems intuitive, I have witnessed seasoned administrators running around in a panic with floppies in hand and trying to remember which servers were running certain services. If a worm exploits a flaw in the sendmail program but the system's MTA is Postfix, why would someone even consider installing a patch or fix on a production server? Know your network to the greatest extent possible.

Tools That Are Helpful in Keeping Systems Patched and Up to Date

Assuming that the systems are now fully documented regarding running services, how is one to know if a program is in need of an update? This used to be a daunting task for the administrator, because it usually meant spending many hours per week perusing hard to navigate vendor/program websites or Internet mailing lists. Fortunately, over the past few years programs have begun to appear designed to help system administrators manage security patches, program updates and hot fixes. I will briefly discuss a few of them.

An administrator in charge of a Windows based network may be familiar with a program called HFNetCheck. This program is written by Shavlik Technologies and checks for the existence of service packs and hot-fixes for the NT/2000 operating system, SQL Server, IIS and Microsoft Internet explorer. The program then helps the user obtain these patches. The HFNetCheck program consults an online resource in order to check for program/system updates. In order to use this program, the computer in question must be connected to the Internet. Another tool is a program called the Microsoft Baseline Security Analyzer (MBSA), which performs similar checks, but offers additional functionality[12]. The Windows NT Resource Kit (NTRK), is also full of programs written to help an NT administrator better manage his or her network and systems.

A Linux system administrator, on the other hand, may be aware of programs such as Red Hat's up2date or Debian's apt-get. Up2date enables system registration, creation of system profiles, and configuration of the way that the local machine interacts with the Red Hat network. After its configuration, it provides the conduit to receive all of the latest operating system and program updates, as well as security patches and bug fixes.[13] Unlike HFNetcheck

[12] Fonseca
[13] Red Hat.com

however, up2date can be configured to automatically update any software the administrator desires.

The apt-get system uses a similar methodology for updating programs. In order to use this system, the administrator would type *apt-get install* and the program name. The program looks for the most recent version of the package and a network connection is opened, the package is downloaded from the correct archive as specified in */etc/apt/sources.list*, dependancies (if any) are resolved, and it is installed. Whole systems and environments can be automatically updated this way.

Do All Vendor Patches Need to Be Applied?

It is important to perform a risk-based analysis before installing a patch or a hot fix. If a patch fixes a hole that can only be exploited from the keyboard of a local server (since the only people who can possibly access that server are trusted, solid employees), and has a history of forcing machines into a kernel panic, a risk analysis might show that installing that patch is more dangerous than not installing it at all.

If after a risk assessment it is decided that a patch must be installed, the administrator needs to make a decision regarding how long to wait before a patch can be installed. One possible scenario might be that a flaw is found in a certain version of Windows NT and can be exploited through a script that is installed by default in IIS 4.0. Your Windows NT based network is running IIS 4.0. The server administrator installed IIS with all of its default options. How long should the administrator wait to install the patch? I have read many sad stories about network and security administrators who installed zero-day patches only to find that when the machine was rebooted (a common event during the patching of a Windows machine), formerly operational services no longer worked. Unfortunately, sometimes a patch may contain code that affects another service on the same machine. Even reputable software companies have released patches that corrupted system or kernel level files and prevented an important server (aren't they all important?) from restarting at all. These issues are important to consider when thinking about mitigating the risk associated with system component upgrades and hot-fixes.

While it is not always possible to have a complete test or staging environment in order to test these bug fixes, it is always important to perform as much testing as possible. If it is not possible to test a new patch on servers at all due to the lack of available test machines, test the patch on a number of workstations that are running on the network. Prepare and train a "tiger team" in advance so that if a virus or worm problem arises, a plan will be in place to minimize and mitigate damage. It is also important to monitor some of the many mailing lists available over the Internet. These lists contain critical information regarding patches and bug fixes. The recent MS03-007 patch for Windows 2000 servers running IIS 5.0

illustrates this point very well. The patch was released in order to fix vulnerability in a Windows component that is used by WebDAV (a set of extensions to HTTP defined in RFC 2518). Some unfortunate administrators found out that in many circumstances, unless the OS was patched to Windows 2000 Service Pack level 3, their servers might not reboot. The same patch also negatively impacted machines running Macromedia Cold Fusion MX. Microsoft corrected this problem a few days later by releasing a "fixed" version of the same patch, but the damage had already been done. Had these administrators held off installing this patch and consulted the mailing lists for three or four days, they would have been aware of these problems and been prepared to more effectively manage these risks.

Conclusion

We will never be able to truly free ourselves from the risks of worms and viruses. Unethical people will never stop trying to exploit our networks and systems. However, by understanding the ways that they can harm our networks with their code, we can put into place processes that hopefully will help make their impact much less severe.

Although Slammer's payload was not destructive, it did cause widespread damage to financial, governmental and commercial networks. Even though administrators will never be able to compete with the speed of such a worm, through proper planning, they can be better prepared to mitigate the risk of such an attack.

Bibliography/References

1. Keyhoe, Brendan P. “The Robert Morris Internet Worm.”Zen and the art of the Internet. January 1992. URL: http://www.swiss.ai.mit.edu/6805/articles/morris-worm.html (18 Mar. 2003)
2. Keyhoe, Brendan P. “The Robert Morris Internet Worm.”Zen and the art of the Internet. January 1992. URL: http://www.swiss.ai.mit.edu/6805/articles/morris-worm.html (18 Mar. 2003)
3. Simenin, VL. “Stopping New Generation of Internet Worms: Mission Impossible?” 6 Mar 2001. URL:http://www.pcflank.com/art18.htm (18 Mar. 2001)
4. Weaver, Nicholas C. “Warhol Worms: The Potential for Very Fast Internet Plagues.” 13 Feb. 2002. URL: http://www.cs.berkeley.edu/~nweaver/warhol.html (18 Mar. 2003)
5. Moore, David *et al.* “The Spread of the Sapphire/Slammer Worm.” Jan. 2003. URL:http://www.cs.berkeley.edu/~nweaver/sapphire/ (19 Mar. 2003)
6. Moore, David *et al.* “The Spread of the Sapphire/Slammer Worm.” Jan. 2003. URL: http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html (20 Mar. 2003)
7. Moore, David *et al.* “The Spread of the Sapphire/Slammer Worm.” Jan. 2003. URL: http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html (20 Mar. 2003)
8. Johnson, Rees. “Protecting Your Data and Your Microsoft SQL Server.” 2 Oct. 2002. URL: http://www.entercept.com/whitepaper/sqlserver/SQLServerSecurity.pdf (20 Mar. 2003)
9. Author Unknown. Document Date Unknown. http://www.sqlsecurity.com/forum/applicationslistgridall.aspx (20 Mar. 2003)
10. Author Unknown. “Elevation of Privilege in SQL Server Web Tasks (Q316333).” Microsoft Security Bulletin MS02-061. 28 Feb. 2003 URL: http://www.microsoft.com/technet/treeview/?url=/technet/security/bulletin/MS02-061.asp (20 Mar. 2003)
11. Staff Writers, ZDNet. Cleaning up after Slammer. 28 Jan. 2003. URL: http://www.zdnet.com.au/newstech/security/story/0,2000024985,20271548,00.htm (20 Mar. 2003)
12. Fonseca, Brian. “Microsoft defends Baseline Security Analyzer tool.” 17 Apr. 2002. URL:http://www.infoworld.com/article/02/04/17/020417hnmsbsa_1.html (21 Mar. 2003)
13. Author Unknown. “Chapter 2. Red Hat Update Agent.” Red Hat Network Basic: User Reference Guide 4.0. Jan. 2003 URL: http://www.redhat.com/docs/manuals/RHNetwork/ref-guide/up2date.html (21 Mar. 2003)