Trends in Viruses and Worms

by Thomas M. Chen, Southern Methodist University

The modern computer *virus* was conceived and demonstrated by Fred Cohen in 1983. Like biological viruses, computer viruses reproduce by attaching to a normal program or document and taking over control of the execution of that program to infect other programs. Early viruses could spread slowly mostly by floppies (such as the 1986 *Brain* virus), but the Internet has made it much easier for viruses to move among computers and spread rapidly. Networks have created a fertile environment for worms, which are related to viruses in their ability to self-replicate but are not attached to other programs. Worms are particularly worrisome as standalone automated programs designed to exploit the network to seek out vulnerable computers. The term *worm* was originated by John Shoch and Jon Hupp during their experiments on mobile software at Xerox PARC in 1979, inspired by the network-based *tapeworm* monster in John Brunner's novel, *The Shockwave Rider*[1]. Shoch and Hupp thought of worms as multi-segmented programs distributed across networked computers.

The Internet increases the vulnerability of all interconnected machines by making it easier for malicious programs to travel between computers by themselves. Recent virus and worm outbreaks, such as the *Blaster* worm in August 2003 and the SQL *Sapphire/Slammer* worm in January 2003, have demonstrated that networked computers continue to be vulnerable to new attacks despite the widespread deployment of antivirus software and firewalls. Indeed, a review of the history of viruses and worms shows that they have continually grown in sophistication over the years. This article highlights a series of significant past innovations in virus and worm technology. The purpose is to show that viruses and worms continue to pose a major risk today and most likely into the future as their creators persist in seeking ways to exploit security weaknesses in networked systems.

Stealth

The earliest viruses attempted to hide evidence of their presence, a trend that continues to today. The 1986 DOS-based *Brain* virus hid itself in memory by simulating all of the DOS system calls that normally detect viruses, causing them to return information that gave the appearance that the virus was not there.

The 2001 *Lion* worm installed a rootkit called *t0rn*, which is designed to make the actions of the worm harder to detect through numerous system modifications to deceive *syslogd* from properly capturing system events (*syslogd* is often used to detect worm activity)[2]. More recently, viruses and worms have attempted to hide by actively attacking antivirus software on the infected computer (refer to the section "Armoring").

Social Engineering

The 1987 *Christma Exec* virus was an early example of social engineering, spreading by e-mail among IBM mainframes. An arriving message tricks the user into executing the virus by promising to draw a Christmas tree graphic. The virus does produce a Christmas card graphic on the computer display (drawn using a scripting language called *Rexx*) but sends a copy of itself in the user's name to that user's list of outgoing mail recipients. The recipients believe the e-mail is from the user, so they are more likely to open the e-mail.

Social engineering continues to be common practice in today's viruses and worms, particularly those spread by e-mail. In January 1999, the *Happy99/Ska* worm/Trojan horse hybrid spread by e-mail with an attachment called **`Happy99.exe`**[3]. When the attachment was executed, it displayed fireworks on the screen to commemorate New Year's Day, but secretly modified the **`WSOCK32.DLL`** file (the main Windows file for Internet communications) with a Trojan horse program that allowed the worm to insert itself into the Internet communications process. Every e-mail sent by the user generated a second copy without any text but carried the worm to the same recipients.

The 1999 *PrettyPark* worm propagated as an e-mail attachment called **`Pretty Park.exe`**. The attachment is not explained, but it bears the icon of a character from the television show, *South Park*. If executed, it installs itself into the Windows System folder and modifies the Registry to ensure that it runs whenever any **`.EXE`** program is executed. In addition, the worm e-mails itself to addresses found in the Windows Address Book. It also mails some private system data and passwords to certain *Internet Relay Chat* (IRC) servers. Reportedly, the worm also installs a backdoor to allow a remote machine to create and remove directories, and send, receive, and execute files.

In February 2001, the *Anna Kournikova* virus demonstrated social engineering again, pretending to carry a JPG picture of the tennis player. If executed, the virus e-mails a copy of itself to all addresses in the Outlook address book.

In March 2002, the *Gibe* worm spread as an attachment in an e-mail disguised as a Microsoft security bulletin and patch. The text claimed that the attachment was a Microsoft security patch for Outlook and Internet Explorer. If the attachment is executed, it displays dialog boxes that appear to be patching the system, but a backdoor is secretly installed on the system.

Macro Viruses

The *Concept* virus was the first macro virus, written for Word for Windows 95. The vast majority of macro viruses are targeted to Microsoft Office documents that save macro code within the body of documents. Macro viruses have the advantages of being easy to write and independent of computing platform. However, macro viruses are no longer widespread after people have become more cautious about using the Office macro feature.

Mass E-Mailers

In March 1999, the *Melissa* macro virus spread quickly to 100,000 hosts around the world in three days, setting a new record and shutting down e-mail for many organizations using Microsoft Exchange Server[4]. It began as a newsgroup posting promising account names and passwords for erotic Web sites. However, the downloaded Word document actually contained a macro that used the functions of Microsoft Word and the Microsoft Outlook e-mail program to propagate. Up to that time, it was widely believed that a computer could not become infected with a virus just by opening e-mail. When the macro is executed in Word, it first checks whether the installed version of Word is infectable. If it is, it reduces the security setting on Word to prevent it from displaying any warnings about macro content. Next, the virus looks for a certain Registry key containing the word "Kwyjibo" (apparently from an episode of the television show, *The Simpsons*). In the absence of this key, the virus launches Outlook and sends itself to 50 recipients found in the address book. Additionally, it infects the Word **`NORMAL.DOT`** template using the Microsoft *Visual Basic for Applications* (VBA) macro auto-execute feature. Any Word document saved from the template would carry the virus.

In June 1999, the *ExploreZip* worm appeared to be a WinZip file attached to e-mail but was not really a zipped file[5]. If executed, it appears to display an error message, but the worm secretly copies itself into the Windows Systems directory or loads itself into the Registry. It sends itself via e-mail using Outlook or Exchange to recipients found in unread messages in the inbox. It monitors all incoming messages and replies to the sender with a copy of itself.

In May 2000, the fast-spreading *Love Letter* worm demonstrated a social engineering attack[6]. It propagated as an e-mail message with the subject "I love you" and text that encourages the recipient to read the attachment. The attachment is a Visual Basic script that could be executed with Windows Script Host (present if the computer has Windows 98, Windows 2000, Internet Explorer 5, or Outlook 5). Upon execution, the worm installs copies of itself into the Windows System directory and modifies the Registry to ensure that the files are run when the computer starts up. The worm also infects various types of files (for example, **`.VBS`**, **`.JPG`**, **`.MP3`**, etc.) on local drives and networked shared directories. If Outlook is installed, the worm e-mails copies of itself to addresses found in the address book. In addition, the worm makes a connection to IRC and sends a copy of itself to anyone who joins the IRC channel. The worm has a password-stealing feature that changes the startup URL in Internet Explorer to a Website in Asia. The Website downloads a Trojan horse designed to collect various passwords from the computer.

In 2002, 90 percent of the known viruses were mass e-mailers. Two of the most prevalent ones, *Bugbear* and *Klez,* began a trend of carrying their own *Simple Mail Transfer Protocol* (SMTP) engines. Although e-mail continues to be the most common infection vector, recent worms have been exploring new vectors (see the section "New Infection Vectors").

In addition, mail servers are becoming more powerful in their capabilities to detect and filter malicious code. For these reasons, mass e-mailing may decline as an infection vector for future viruses.

Polymorphism

Polymorphism is based on the simpler idea of encryption, which makes a virus harder to detect by antivirus software scanning for a unique virus signature (byte pattern). Encryption attempts to hide a recognizable signature by scrambling the virus body. To be executable, the encrypted virus is prepended with a decryption routine and encryption key. However, encryption is not effective because the decryption routine remains the same from generation to generation, although the key can change, scrambling the virus body differently. Antivirus scanners can detect a sequence of bytes identifying a specific decryption scheme.

Polymorphic viruses permute continuously to avoid detection by antivirus scanning[7]. The earliest polymorphic virus might have been a virus found in Europe in 1989. This virus replicated by inserting a pseudorandom number of extra bytes into the decryption algorithm, preventing any common sequence of more than a few bytes between two successive infections. Polymorphism became practical when a well-known hacker, *Dark Avenger,* developed a user-friendly *Mutation Engine* program to provide any virus with variable encryption. With a static signature so small, the risk of false positives by antivirus scanners became very high. Other hackers soon followed with their own versions of so-called mutation engines. The 1995 *Pathogen* and *Queeg* viruses were polymorphic DOS file-infecting viruses produced by Black Baron's *Simulated Metamorphic Encryption enGine* (SMEG)[7].

Blended Attacks

The famous 1988 *Morris* worm was the first to use a combination of attacks (or blended attacks) to spread quickly to 6000 UNIX computers in a few hours (10 percent of the Internet at that time)[8].

- It captured the password file and ran a password-guessing program on it using a dictionary of common words.
- It exploited the debug option in the UNIX *sendmail* program, allowing it to transfer a copy of itself.
- It carried out a buffer overflow attack through a vulnerability in the UNIX *fingerd* program.

In May 2001, the *Sadmind/IIS* worm spread by targeting two separate vulnerabilities on two different operating systems. It first exploited a buffer overflow vulnerability in Sun Solaris systems and installed software to carry out an attack to compromise Microsoft *Internet Information Services* (IIS) Web servers.

The July 2001 *Sircam* worm uses two ways to propagate. First, it e-mails itself as an attachment using its own SMTP engine, and if the attachment is executed, e-mails a copy of itself to addresses found in the Windows address book. Second, it spreads by infection of unprotected network shares.

In September 2001, *Nimda* raised new alarms by using five different ways to spread to 450,000 hosts within the first 12 hours[9]. *Nimda* seemed to signal a new level of worm sophistication.

- It found e-mail addresses from the computer Web cache and default *Messaging Application Programming Interface* (MAPI) mailbox. It sent itself by e-mail with random subjects and an attachment named **`readme.exe`**. If the target system supported the automatic execution of embedded MIME types, the attached worm would be automatically executed and infect the target.
- It infected Microsoft IIS Web servers, selected at random, through a buffer overflow attack called a *unicode* Web traversal exploit.
- It copied itself across open network shares. On an infected server, the worm wrote *Multipurpose Internet Mail Extensions* (MIME)-encoded copies of itself to every directory, including network shares.
- It added JavaScript to Web pages to infect any Web browsers going to that Website.
- It looked for backdoors left by previous *Code Red II* and *Sadmind* worms.

Armoring

In November 2002, the *Winevar* worm was an example of an "armored" worm that contained special code designed to disable antivirus software using a list of keywords to scan memory to recognize and stop antivirus processes and scan hard drives to delete associated files[10].

Klez and *Bugbear* are recent examples of worms that attack antivirus software by stopping active processes and deleting registry keys and database files used by popular antivirus programs. The 2003 *Fizzer* and *Lirva* worms also attempt to disable antivirus software.

Dynamic Software Updates

In October 2000, the *Hybris* worm propagated as an e-mail attachment[11]. It connected to the **`alt.comp.virus`** newsgroup to receive encrypted plug-ins (code updates). The method is sophisticated and potentially very dangerous, because the worm payload (destructive capability) can be modified dynamically.

The 2003 *Lirva* worm attempted to connect to a Website on **`web.host.kz`** to download BackOrifice, a notorious remote-access software package that gives complete control to a remote attacker. It also attempted to download another unknown file that was not found on the Website.

This technique was given an interesting twist by the *Welchia* or *Nachi* worm, which began spreading on August 18, 2003, soon after the *Blaster* worm. Apparently, its creator intended *Welchia* as a "good" worm to remove *Blaster*. It attempted to download and install a fix for *Blaster* from a Microsoft Website.

New Infection Vectors

The Linux *Slapper* worm, appearing in September 2002, was among the first to exploit *peer-to-peer* (P2P) technology[12]. It spread to Linux computers by exploiting the long *Secure Sockets Layer 2* (SSL2) key argument buffer overflow in the *libssl* library, used by the *mod_ssl* module of the Apache 1.3 Web server. When the worm infects a new machine, it binds to *User Datagram Protocol* (UDP) port 2002 and becomes part of a P2P network. The parent of the worm on the attacking machine sends to its offspring the list of all hosts on the P2P network and broadcasts the address of the new worm on the network. Then periodic updates to the host list are exchanged between machines on the network. The new worm also scans the network for other vulnerable machines, sweeping randomly chosen class B networks.

In March 2003, the *AimVen* worm spread by the *America OnLine Instant Messager* (AIM) by modifying the AIM program. Whenever an **.EXE** file is sent through AIM, the worm overwrites the file with a copy of itself.

The *Fizzer* worm discovered in May 2003 is a mass e-mailer that includes its own SMTP engine like *Klez* and *Bugbear*. It also tries to spread via *KaZaa*, a popular P2P file-sharing application, and shared directories.

The 2003 *Lirva* worm, named after the singer, Avril Lavigne, is a mass e-mailer taking advantage of the same MIME header exploit as *Badtrans* and *Klez*, but also tries to spread by IRC, "I seek You" (ICQ), *KaZaa,* and open network shares[13].

Data-Stealing Payloads

Most fast-spreading worms in the past have not carried destructive payloads. Instead, they have tended to appear to be proof-of-concepts to demonstrate a particular security weakness. Some worms, though, such as *Code Red,* have installed *Denial-of-Service* (DoS) agents or backdoors on infected machines. Recently worms have begun to carry keyloggers and password-stealing Trojans in their payloads.

The 2003 *Fizzer* worm includes a keystroke logging Trojan horse that stores the data in an encrypted file. It establishes its own accounts on IRC and AIM to wait for instructions from the virus writer, who could conceivably fetch the keystrokes data.

The 2003 *Lirva* worm e-mails cached Windows dialup networking passwords to the virus writer, and e-mail random **.TXT** and **.DOC** files to various addresses.

Bugbear installs a keystroke logging tool into the Windows System folder that e-mails the keystrokes data to preprogrammed addresses[14]. It listens on port 36794 for commands from a remote hacker.

Fast and Furious Worms

A particularly worrisome new trend is extremely fast worms targeted to specific (usually Windows-related) vulnerabilities that might saturate their target population within a few hours or even less than an hour. These worms tend to be simpler and targeted to single rather than multiple vulnerabilities, in order to be highly efficient in their probing for other vulnerable machines.

The first example might be the *Code Red* worm, which actually appeared in three different versions[15]. The first version of *Code Red I* appeared on July 12, 2001, targeted to a buffer overflow vulnerability in Microsoft IIS Web servers. However, a programming error in its pseudorandom address generator caused each worm copy to probe the same set of IP addresses and prevented the worm from spreading quickly. A week later on July 19, a second version of *Code Red I* with the programming error apparently fixed was able to infect more than 359,000 servers within 14 hours. At its peak, the worm was infecting 2000 hosts every minute. A more complex and dangerous *Code Red II* targeted to the same IIS vulnerability appeared on August 4.

More recently, the *Structured Query Language* (SQL) *Sapphire/ Slammer* worm appeared on January 25, 2003, targeted to Microsoft SQL Server machines not running *Service Pack 3* (SP3), such as SQL Server 2000 and *Microsoft Desktop Engine* (MSDE) 2000[16]. It reportedly infected 90 percent of vulnerable hosts within 10 minutes (about 120,000 servers)[17]. The spreading rate was surprisingly fast and resulted in DoS effects (network outages and high packet loss) due to traffic overloading servers and routers. In the first minute, the infection doubled every 8.5 seconds, and hit a peak scanning rate of 55,000,000 scans per second after only 3 minutes. In comparison, *Code Red* infection doubled in 37 minutes (slower but infected more machines). *Slammer* was able to spread so quickly because it appeared to be designed simply for efficient replication. The worm carried no payload and consisted of a single 404-byte UDP packet (including 376 bytes for the worm) that could be sent without having to wait for responses from targeted machines. In contrast, *Code Red* was about 4000 bytes and *Nimda* was 60,000 bytes, and their scanning depended on the time to establish TCP connections to targeted machines. The *Slammer* worm was much more efficient, simply generating copies of itself at the full rate of the infected machine.

Latest Developments

The week of August 12–19, 2003, has been called the worst week for worms in history, seeing *MS Blaster, Welchia* (or *Nachi*), and *Sobig.F* in quick succession. *MS Blaster* or *LovSan* was another fast worm, which appeared on August 12, 2003, targeted to a *Windows Distributed Component Object Model* (DCOM) *Remote Procedure Call* (RPC) vulnerability announced on July 16, 2003[18]. The worm probes for a DCOM interface with RPC listening on TCP port 135 on Windows XP and Windows 2000 PCs. Through a buffer overflow attack, the worm causes the target machine to start a remote shell on port 4444 and send a notification to the attacking machine on UDP port 69.

A *Trivial File Transfer Protocol* (TFTP) "get" command is then sent to port 4444, causing the target machine to fetch a copy of the worm as the file **`MSBLAST.EXE`**. In addition to a message against Microsoft, the worm payload carries a DoS agent (using TCP SYN flood) targeted to the Microsoft Website **`windowsupdate.com`** on August 16, 2003. Although *Blaster* has reportedly infected about 400,000 systems, experts reported that the worm did not achieve near its potential spreading rate because of novice programming.

Six days later on August 18, 2003, the apparently well-intended *Welchia* or *Nachi* worm spread by exploiting the same RPC DCOM vulnerability as *Blaster*. It attempted to remove *Blaster* from infected computers and download a security patch from a Microsoft Website to repair the RPC DCOM vulnerability. Unfortunately, its scanning resulted in a DoS effect on some networks, such as Air Canada's check-in system and the U.S. Navy and Marine Corps computers.

The very fast *Sobig.F* worm appeared on the next day, August 19, 2003, only seven days after *Blaster*[19]. The original *Sobig.A* version was discovered in January 2003, and apparently underwent a series of revisions until the most successful *Sobig.F* variant. Similar to earlier variants, *Sobig.F* spreads among Windows machines by e-mail with various subject lines and attachment names, using its own SMTP engine. The worm size is about 73 kilobytes with a few bytes of garbage attached to the end to evade antivirus scanners. It works well because it grabs e-mail addresses from a variety of different types of files on the infected computer and secretly e-mails itself to all of them, pretending to be sent from one of the addresses. At its peak, *Sobig.F* accounted for 1 in every 17 messages, and reportedly produced over 1 million copies of itself within the first 24 hours. Interestingly, the worm was programmed to stop spreading on September 10, 2003, suggesting that the worm was intended as a proof-of-concept. This is supported by the absence of a destructive payload, although the worm is programmed with the capability to download and execute arbitrary files to infected computers. The downloading is triggered on specific times and weekdays, which are obtained via one of several *Network Time Protocol* (NTP) servers. The worm sends a UDP probe to port 8998 on one of several preprogrammed servers, which responds with a URL for the worm to download. The worm also starts to listen on UDP ports 995–999 for incoming messages, presumably instructions from the creator.

Conclusions

Why does the Internet remain vulnerable to large-scale worm outbreaks? Since at least 1983, the Internet community has understood the risks and mechanics of viruses. The 1988 Morris worm taught the community to be watchful for potentially dangerous worms. Over the years, a variety of antivirus software, firewalls, intrusion detection systems, and other security equipment have been installed. Moreover, the *Computer Emergency Response Team* (CERT) at CMU was established as the first computer security incident response team, which later joined an expansive global coalition of security incident response teams called the *Forum of Incident Response and Security Teams* (FIRST)[20].

Despite our knowledge and infrastructure defenses, many viruses and worms have broken out regularly in the Internet over the years. By some reports, 5 to 15 new viruses and worms are released every day, although a fraction of that number are not released in the wild and most do not spread well. Still, fast-spreading viruses and worms continue to appear with regularity. Outbreaks have become so commonplace that most organizations have come to view them as a routine cost of operation.

The problem is sometimes portrayed as a perpetual struggle between virus writers who keep innovating (as described here) and the antivirus industry, which tries to keep up. However, the problem is actually larger, involving the entire computer industry. Viruses and worms are successful because computers have security vulnerabilities that can be exploited. Clearly, the Internet itself is simply serving its purpose of interconnecting computer systems. The security vulnerabilities exist in the host end systems. Security vulnerabilities continue to exist for many reasons. First, software is often written in an unsecure manner, for example, vulnerable to buffer overflow attacks that are commonly used by worms. Buffer overflow attacks have been widely known since 1995, but this type of vulnerability continues to be found very often (on every operating system.) Second, when vulnerabilities are announced with corresponding software patches, many people are slow to apply patches to their computer for various practical reasons. Weakly protected computers can be compromised, putting the entire community at risk, including secured computers that can still be impacted by the traffic effects of a worm outbreak.

However, there is reason to be hopeful for a solution. Fortunately, worms typically have a weakness of exploiting vulnerabilities that have been known for some time. Worm writers do not invent new exploits for the simple reason that they want to ensure that their worm will spread after it is released. For example, the *Code Red I* worm took advantage of a buffer overflow vulnerability in Microsoft IIS servers that had been known for a month. The *Nimda* worm exploited a unicode Web traversal vulnerability in Microsoft IIS servers that was published a year earlier. The SQL *Slammer/Sapphire* worm exploited a buffer overflow vulnerability in Microsoft SQL servers that had been known for six months. The recent *Blaster* worm exploited a Windows DCOM RPC vulnerability announced two months earlier. Watching for probing activity attempting to exploit known vulnerabilities could help detect and block worm outbreaks at an early stage. Ideas for automatic detection and quarantine of new epidemics is attracting research[21].

Aside from technological considerations, an important issue is accountability. The most obvious parties to hold liable are the virus creators, but it has been observed many times that few virus writers have been prosecuted, and sentences have tended to be light. The author of the 1988 Internet worm, Robert Morris, was sentenced to three years of probation, 400 hours of community service, and a $10,000 fine.

Chen Ing-hau was arrested in Taiwan for the 1998 *Chernobyl* virus, but he was released when no official complaint was filed. Onel de Guzman was arrested for writing the 2000 *LoveLetter* virus, which resulted in $7 billion of damages, but he was released because of the lack of relevant laws in the Philippines. Jan De Wit was sentenced for the 2001 *Anna Kournikova* virus to 150 hours of community service. David L. Smith, creator of the 1999 *Melissa* that caused at least $80 million of damages, was sentenced to 20 months of custodial service and a $7500 fine.

It is notoriously difficult to trace a virus or worm to its creator from analysis of the code, unless inadvertent clues are left in the code. In addition, cases are difficult to prosecute, and malicious intention (as opposed to just recklessness) is difficult to prove. Moreover, long prison sentences have been perceived as overly harsh for arrested virus creators, who have tended to be teenagers and university students. In addition, in the absence of a serious legal deterrent, the general perception persists that virus creators can easily avoid the legal consequences of their actions. Perhaps to address this problem, authorities have been diligently investigating the creators of *Blaster* and *Sobig*. So far, a teenager, Jeffrey Lee Parson, has been arrested for writing the *Blaster.B* variant, a slight modification of the original *Blaster*. Soon afterward, Dan Dumitru Ciobanu was arrested in Romania for writing the *Blaster.F* variant.

Some have argued wishfully that software vendors should be held financially liable for damages resulting from the security vulnerabilities in their products. The assumption is that accountability would increase motivation to write and sell more secure software, a solution that would result in a less inviting environment for viruses and worms. So far, software vendors have managed to acknowledge their role but avoid accountability.

References

[1] J. Shoch and J. Hupp, "The 'worm' programs—early experience with a distributed computation," *Communications of ACM,* Volume 25, pp. 172–180, March 1982.

[2] A. Kasarda, "The Lion worm: king of the jungle?" SANS reading room, **http://www.sans.org/rr**

[3] CERT incident note CA-1999-02, "Happy99.exe trojan horse," **http://www.cert.org/incident_notes/IN-99-02.html**

[4] CERT advisory CA-1999-04, "Melissa macro virus," **http://www.cert.org/advisories/CA-1999-04.html**

[5] CERT advisory CA-1999-06, "ExploreZip trojan horse program," **http://www.cert.org/advisories/CA-1999-06.html**

[6] CERT advisory CA-2000-04, "Love letter worm," **http://www.cert.org/advisories/CA-2000-04.html**

[7] D. Harley, R. Slade, and R. Gattiker, *Viruses Revealed,* Osborne/McGraw-Hill, 2001.

[8] E. Spafford, "The Internet worm program: an analysis," *ACM Computer Communications Review,* Volume 19, pp. 17–57, January 1989.

[9] CERT advisory CA-2001-26, "Nimda worm,"
http://www.cert.org/advisories/CA-2001-26.html

[10] Virus Bulletin, "W32/WineVar,"
http://www.virusbtn.com/resources/viruses/winevar.xml

[11] CERT incident note IN-2001-02, "Open mail relays used to deliver Hybris worm,"
http://www.cert.org/incident_notes/IN-2001-02.html

[12] F-Secure, "F-Secure virus descriptions: Slapper,"
http://www.f-secure.com/v-descs/slapper.shtml

[13] Symantec Security Response, "W32.lirva.C@mm,"
http://securityresponse.symantec.com/avcenter/venc/data/w32.lirva.c@mm .html

[14] Sophos, "W32/Bugbear-A,"
http://www.sophos.com/virusinfo/analyses/w32bugbeara.html

[15] H. Berghel, "The Code Red worm," *Communications of ACM,* Volume 44, pp. 15–19, December 2001.

[16] CERT advisory CA-2003-04, "MS-SQL server worm,"
http://www.cert.org/advisories/CA-2003-04.html

[17] D. Moore, et al., "The spread of the Sapphire/Slammer worm,"
http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html

[18] CERT advisory CA-2003-20, "W32/Blaster worm," Aug. 11, 2003,
http://www.cert.org/advisories/CA-2003-20.html

[19] Symantec Security Response, "W32.Sobig.F@mm,"
http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm .html

[20] Forum of Incident Response and Security Teams (FIRST),
http://www.first.org

[21] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet quarantine: requirements for containing self-propagating code," IEEE Infocom 2003, San Francisco, April 2003.

THOMAS M. CHEN holds BS and MS degrees in electrical engineering from MIT, and a PhD in electrical engineering from the University of California, Berkeley. From 1989 to 1997, he worked on ATM networking research at GTE Laboratories (now Verizon). He is currently an Associate Professor in the Department of Electrical Engineering at SMU in Dallas, Texas. He is the associate editor-in-chief of *IEEE Communications Magazine,* a senior editor of *IEEE Network,* an associate editor of *ACM Transactions on Internet Technology,* and founding editor of *IEEE Communications Surveys.* He is the coauthor of *ATM Switching Systems* (Artech House, 1995). E-mail: **tchen@engr.smu.edu**

Meanwhile, the nature of current worms and viruses is also changing considerablyâ€”a growing number of them uses instant messaging (IM) to replicate, and worms and viruses that target handheld computing devices are also becoming more prevalent. Bots and botnets pose very elevated levels of risk, risk that needs to be controlled through a variety of security countermeasures. Examples: Worms and Zombie programs. A virus is a piece of software that can â€œinfectâ€ other programs by modifying them. The modification includes a copy of the virus program, which then can go to infect other programs.Â Most viruses carry out their work in a manner that is specific to6a particular operating system and in some cases specific to a particular hardware platform. Thus they are designed to take advantage of the details and weaknesses of particular systems. iv. A virus can be prepended or postpended to an executable program, or it can be embedded in some other fashion. The key to itâ€™s operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.