



realtimepublishers.com™

*The Definitive Guide™ To*

# Controlling Malware, Spyware, Phishing, and Spam

**McAfee®**  
Proven Security™

*Dan Sullivan*

Chapter 3: Viruses, Worms, and Blended Threats.....	46
Evolution of Viruses and Countermeasures.....	46
The Early Days of Viruses.....	47
Beyond Annoyance: The Proliferation of Destructive Viruses .....	48
Wiping Out Hard Drives—CIH Virus .....	48
Virus Programming for the Masses 1: Macro Viruses.....	48
Virus Programming for the Masses 2: Virus Generators.....	50
Evolving Threats, Evolving Countermeasures .....	51
Detecting Viruses.....	51
Radical Evolution—Polymorphic and Metamorphic Viruses .....	53
Detecting Complex Viruses .....	55
State of Virus Detection.....	55
Trends in Virus Evolution.....	56
Worms and Vulnerabilities .....	57
Early Worms .....	57
Implementation Techniques and Consequences .....	57
Sobig.....	58
MyDoom.....	58
Sdbot .....	59
SQL Slammer.....	60
Increasing Malevolence .....	62
The New Frontier—Blended Threats.....	63
Multiple Methods of Attack.....	63
Multiple Methods of Transmission.....	64
Multiple Methods of Control .....	64
Summary .....	65

## Copyright Statement

© 2005 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at [info@realtimepublishers.com](mailto:info@realtimepublishers.com).

[**Editor's Note:** This eBook was downloaded from Content Central. To download other eBooks on this topic, please visit [http://www.realtimepublishers.com/contentcentral/.](http://www.realtimepublishers.com/contentcentral/)]

## Chapter 3: Viruses, Worms, and Blended Threats

Viruses, worms, and blended threats are all examples of malicious code collectively known as *malware*. Malicious programs have existed since (at least) the early 1980s with the advent of personal computing. Since then, viruses, worms, and related programs have evolved rapidly, often in response to new opportunities presented by advances in networking or application features. Other times, virus writers are forced to adapt to avoid detection by ever more sophisticated detection techniques and countermeasures.

This chapter examines the history of some of the most common types of malware: viruses and worms. Both types of malware can succeed only when they can replicate and spread without detection. Much of the effort needed to deploy a virus goes into disguising itself to avoid detection. Worms similarly try to hide themselves, but variants exist that have opted to remain in the open and propagate rapidly and in large numbers to survive and spread. There is no single programming technique or stealth strategy deployed by these prominent forms of malware; rather, like their biological namesakes, they have adopted and survived using a variety of techniques.

In addition to using multiple methods to ensure their survival, malicious programs have evolved to become more than a single virus or worm and are now often a collection of multiple pieces of malware operating together to compromise computing platforms. These multiple-threat programs, known as blended threats, are common today. This trend is driven, in part, by emerging uses of malware. The motives for writing and deploying malware have also changed over the past two decades as the economic dimension of malware has emerged to provide one of the most powerful incentives for creating malicious code.

### Evolution of Viruses and Countermeasures

Technically, a virus is a program that can replicate itself using another program running on a host. These programs are usually also malicious, carrying out destructive acts, including:

- Deleting files
- Corrupting files
- Destabilizing OS runtime environments
- Vandalizing platforms (for example, displaying messages about the virus compromise)
- Spreading spam
- Capturing personally identifying information

This list is by no means an exhaustive inventory of malicious acts that viruses can perpetrate, but it is representative of two notable facets of malware:

- The potential loss due to a virus infection ranges from a minor annoyance, as is the case with simple vandalism, to significant loss or disruption in operations, as in the case of stealing personal information.
- The incentives of virus writing were once limited to proving one's coding prowess and earning the respect of other attackers, but now include economic gain from identity theft and other fraudulent acts.

To understand both the technical evolution of viruses and the changing social and economic incentives for their development, it is worth starting at the beginning with computer viruses.

### ***The Early Days of Viruses***


Early viruses were, by today's standards, simplistic and benign. They would infect the boot sector of a floppy disk and cause minor damage, such as occupying small amounts of storage space and displaying annoying messages. The basic goal was to get the virus loaded into memory when the legitimate program was executed and then to infect other programs.

For example, the Elk Cloner virus, written in 1982 by a 15-year-old high school student is generally regarded as the first computer virus outside a research institution (where predecessors of viruses and worms were experimented with since the 1960s). The virus spread by sharing floppy disks. Once activated, the virus would print a poem about "Cloner" affecting the user's machine.

The Brain virus, the first known virus to infect IBM-compatible computers, was designed to prevent the pirating of the author's software. Infected machines would display the message:

"Welcome to the Dungeon (c) 1986 Basit \* Amjad (pvt) Ltd. BRAIN COMPUTER SERVICES 730 NIZAM BLOCK ALLAMA IQBAL TOWN LAHORE-PAKISTAN PHONE: 430791,443248,280530. Beware of this VIRUS.... Contact us for vaccination..."

The purported intent was to prevent users from spreading illegal copies of their program.

 For more information about the Brain virus and the author's reaction to the unintended notoriety, see <http://www.brain.net.pk/aboutus.htm>.

These viruses had few options for spreading. Floppy disk sharing is a well-known method, but bulletin boards were also used. Games and other programs shared on bulletin boards could quickly spread viruses depending on the popularity of the infected program.

Elk Cloner and Brain are examples of parasitic viruses; they used hosts to replicate but did not damage the host systems. Things turned decidedly ugly in the 1990s with the 10,000 DOS-based viruses in existence by 1996, including some truly malevolent viruses.

## **Beyond Annoyance: The Proliferation of Destructive Viruses**

The late 1990s witnessed the wide spread of destructive viruses that propagated more rapidly than early viruses.

### **Wiping Out Hard Drives—CIH Virus**

A destructive virus known as CIH spread through Asia in 1998 after a hard-disk manufacturer shipped a firmware update infected with the virus. The virus overwrites the first 1024 kilobytes of the boot drive, often damaging the partition table; in some PCs, another part of the virus could overwrite the BIOS. The virus took advantage of the fact that the Portable Executable File Format used in the Windows 9x OS often left files with blank space in executable files. The malicious code was essentially hidden in this empty area of the file so as not to change the file size.

Coding a program such as CIH requires in-depth knowledge of OSs, file systems, and systems programming, as well as the ability to code in low-level programming languages, such as assembly language. When viruses were written in assembly language, there was a natural barrier to entry; not every virus programmer want-to-be could design and implement malicious code. Unfortunately, like other technical advances, evolutionary features of desktop applications could be used for both productive and destructive ends.

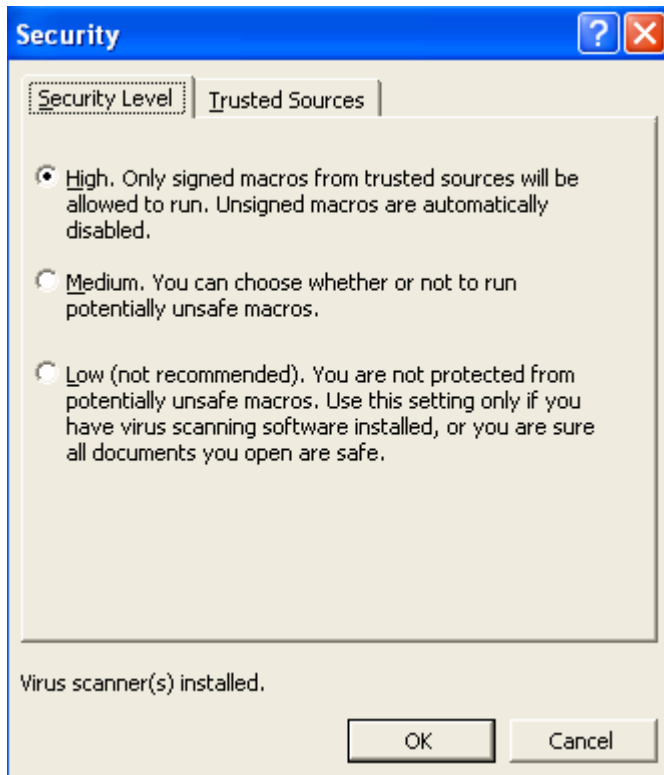
### **Virus Programming for the Masses 1: Macro Viruses**

Until the late 1990s, most viruses spread by attaching themselves to executable programs. By that time, however, advances in desktop applications were providing new methods of spreading (technically, these are known as *vectors*). Visual Basic for Applications (VBA) is a simple programming language embedded in desktop applications such as Microsoft Word and Microsoft Excel to allow users to program macros. Although the intended use was to automate repetitive tasks and improve productivity, virus writers discovered that the lax security features, ease of programming, and rich set of commands made VBA an ideal vector for the next generation of viruses.

Some macro viruses are trivially simple. The Word macro virus, BREEDER, for example, checks a document or template as it is opened, and if it does not have an AutoOpen module, it replicates itself to the file. VONANBUG is another, more malicious, macro virus. When this one executes, it disables Word menu functions, including the ability to view macros. If the user were to press Alt-F11 to view the macro, it would delete files with extensions that indicated graphic, sound, archive, and Web files. In addition to infecting open document files, it infects the template file, normal.dot.

Macro viruses can infect any desktop application that supports macros, not just Microsoft Word. Several macro viruses infect Microsoft Excel with trigger infections that use Auto\_Open routines. The TRASHER.D macro virus, for instance, attempts to delete files in antivirus program directories, and if the day of the month is the 1<sup>st</sup>, 9<sup>th</sup>, or 23<sup>rd</sup>, the program will modify the autoexec.bat file to format the hard drive.

Macro viruses and related macro worms are easy to develop, but application developers have included some security features to limit the risk of these threats. Microsoft applications, for example, allow users to specify a trust level for running macros (see Figure 3.1). In some cases, a high trust level will prevent legitimate macros from running. By lowering the trust threshold to medium, a user can choose which macros to run and which to terminate. This setup is a reasonable balance for some users; others may be unfamiliar with the nature of macros and allow any macro to run when prompted for a decision about executing the embedded code. The flexibility of this type of security feature is certainly welcome but does not eliminate the need to scan all documents to ensure malicious code is not reaching the desktop.



**Figure 3.1:** Desktop applications, such as Microsoft Word, are now employing more security controls to limit the risk of macro viruses.

Macro viruses still require some basic programming skill and understanding of the underlying OS. The bar for creating viruses has been lowered even further with the advent of virus generators.

## Virus Programming for the Masses 2: Virus Generators

Virus generators are programs that create viruses according to a user's specification. These tools, which are freely available on the Internet, enable virtually anyone with the skill to download a file and work with a graphical user interface (GUI) to create a virus. At first glance, the availability of this software is frightening. Fortunately, the code created by these generators is easily detected by antivirus software. As long as there is widespread use of antivirus programs, the threat from "script kiddies" armed with virus generators is controllable.

Computer viruses have evolved to take on a variety of forms from handcrafted assembly language routines to generated viruses. Virus writers have exploited new technologies, such as application macros, to spread their malware; at the same time, antivirus developers have created more effective countermeasures to keep pace with the malware writers. As the McAfee World Virus Map in Figure 3.2 shows, virus infections are a global phenomenon—and not one that is likely to disappear soon.

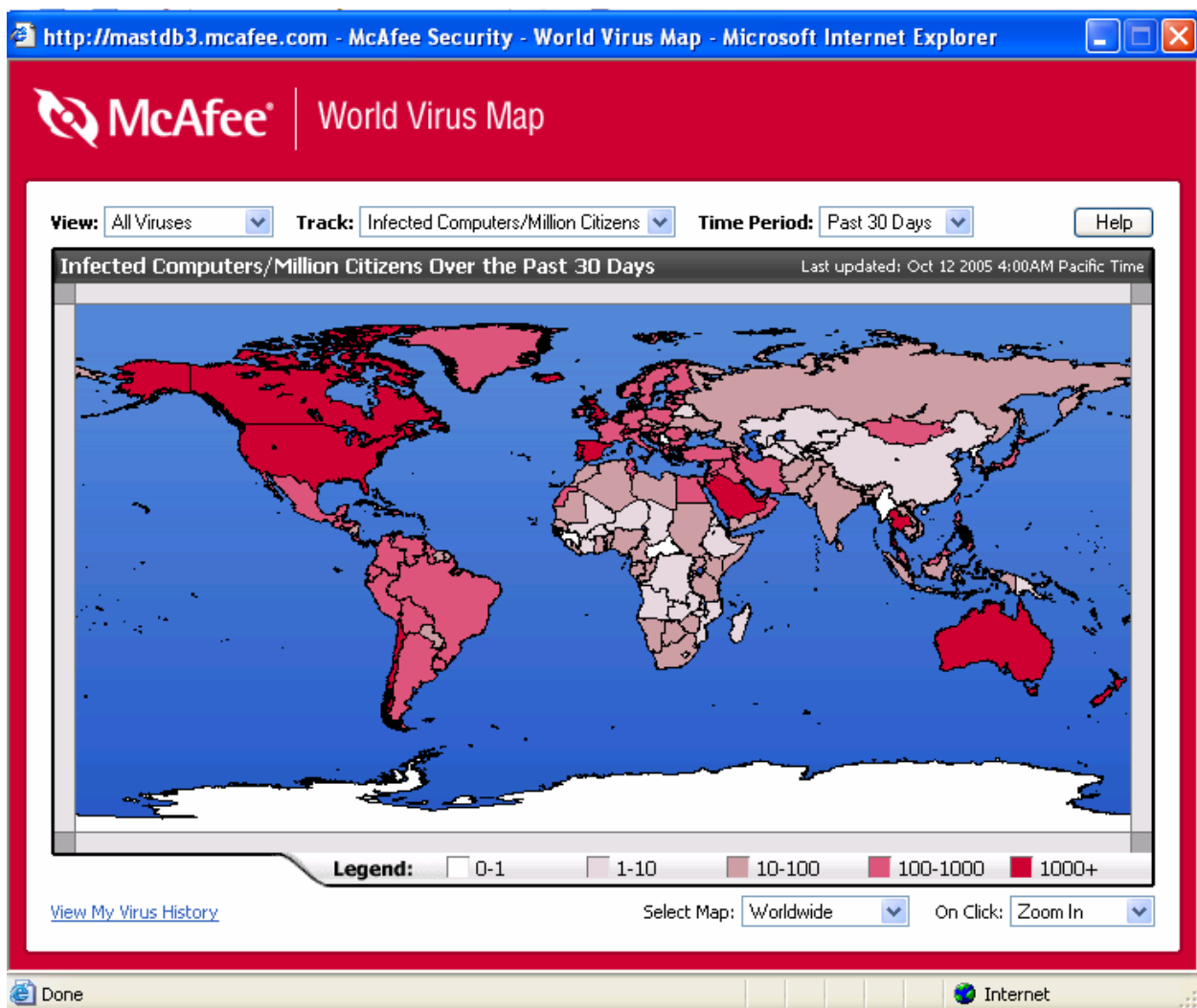


Figure 3.2: Computer viruses are a global problem; North America, Europe, and Australia are most infected.



## ***Evolving Threats, Evolving Countermeasures***

Computer security is often described as a cat and mouse game. Hackers, virus writers, and cyber-vandals exploit vulnerabilities in systems; developers and systems administrators fix flaws in software, change configurations, or deploy safeguards such as anti-virus software. The attackers go back to the drawing board (or keyboard in this case) to find other vulnerabilities that are exploited, causing the security professionals to respond...and so it goes. Although this wording presents a simplistic picture of computer security (security professionals do much more than wait around to respond to threats), it provides a useful framework for understanding the evolution of antivirus countermeasures.

## **Detecting Viruses**

Viruses must remain undetected long enough to propagate in order to succeed in whatever goal they have been designed for. In the early days of DOS-based PCs unprotected with antivirus software, the task was relatively straightforward. If a virus could infect a file or the boot sector on the disk, not take up too much room in RAM, and not change the size of an infected file, chances were good that the virus would replicate. The widespread use of antivirus software changed the playing field.

### ***Early Signature-Based Detection***

Early antivirus programs were essentially string matching programs. When a virus was discovered, it was analyzed to determine the sequence of machine instructions that constituted the program. Antivirus developers would find a unique sequence of instructions that were always present in the virus (and presumably not in non-virus files). That sequence constituted the virus signature that was added to the database of signatures maintained by the antivirus program.

During a virus scan, the antivirus program would open all files and run a pattern-matching algorithm over the file. If a signature was detected in the file, it was infected but otherwise was virus free. Needless to say, this process made virus detection relatively easy (at least by today's standards) until the number of viruses grew to outpace the ability of early programs to effectively scan all files.

Antivirus designers responded by taking advantage of some characteristics of viruses to improve scanning performance, including:

- Most early viruses were relatively small, using less than 8KB of code
- Viruses tended to be located at the beginning or end of a file
- Viruses infect at the entry point of a program, which is the first instruction executed by a program; by analyzing the entry point instruction, virus scanners could detect where the flow of control is transferred, which, in the case of an infected file, would be the location of the virus code

These observations allowed antivirus designers to vastly improve performance over full-file scanning as well as limit the length of scans, focusing on entry points and other techniques derived from the analysis of typical virus structure and function.

Although it is almost a decade old, Carey Nachenberg's paper "Computer Virus Anti-Virus Coevolution" is still an excellent overview of the early approaches to virus detection. The paper is available at [http://crypto.stanford.edu/cs155/virus\\_antivirus\\_coevolution.pdf](http://crypto.stanford.edu/cs155/virus_antivirus_coevolution.pdf).

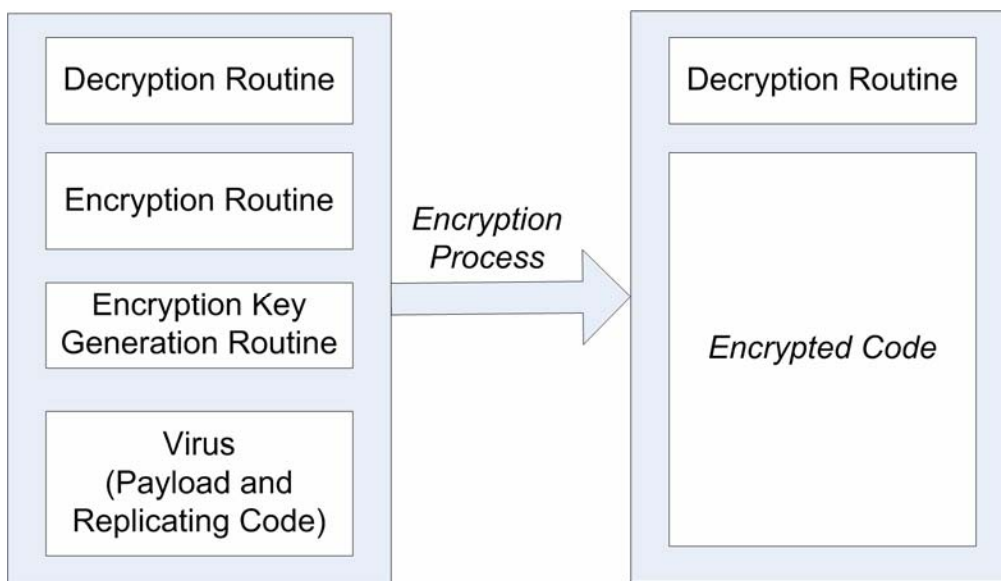
With antivirus scanning keeping up with the volume of viruses, virus writers had to turn to new techniques to avoid detection.

### **Avoiding Detection with Encryption**

The next step in the evolution of viruses was the introduction of encryption routines. This functionality allowed viruses to hide themselves until they were executed. The process works as follows:

- The virus writer creates a virus and embeds encryption and decryption routines.
- When the virus replicates, it encrypts the virus using a different encryption key from the previous generations. This method allows each replicated version to appear different from the other versions.
- The replicated virus continues to carry with it the decryption routine that is executed when the virus is activated.

Traditional virus detection signatures did not work at first because the viruses were encrypted while stored in a file and only decrypted when executed. Fortunately, as Figure 3.3 shows, the decryption routine, although small, was still unencrypted and could be used to identify viruses.



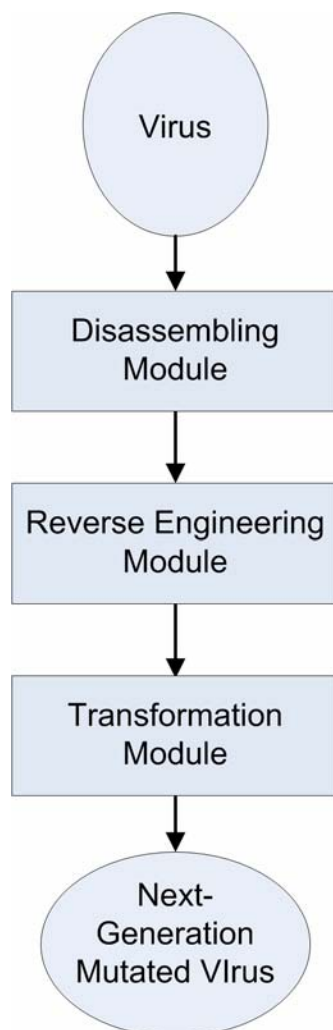
**Figure 3.3:** Even with encryption, enough of the virus program remains unencrypted to allow for signature-based detection.

The one drawback of this approach is that decryption routines can be quite short. Short patterns are more likely to be found in files than long patterns. For example, "mai" is more likely to be found in a randomly selected word document than "virus infected email." The short patterns led to an increase in false positives, which was easily countered by a technique known as x-raying in which the virus is decrypted and the contents examined for known virus signatures. The simple encryption techniques used with viruses made the decryption process relatively straightforward.

## Radical Evolution—Polymorphic and Metamorphic Viruses


The basic signature-based model of virus detection worked for years, even withstanding challenges in the volume of viruses and the use of encryption to hide virus code. The most radical change up to that point in the virus/antivirus co-evolution came in 1992 with the introduction of mutation engines.

As the decryption routine was the Achilles heel of virus writers, they needed a way to mask that code. Encryption would not work because the routine must be in its decrypted form to work. Encryption was not used to keep the virus confidential (the usual reason for encrypting) but to prevent it from being identified by signature-based antivirus detection. An alternative method of masking the code was to change the sequence of instructions in such a way that still maintained the essential functionality of the virus. Mutation engines consist of three parts: a disassembler, a reverse engineering module, and a transformation module (see Figure 3.4).



**Figure 3.4:** A mutation engine uses a three-stage process to analyze a virus and transform it into a functionally equivalent but structurally different next-generation version of the virus.

The job of the disassembler is to identify the programming instructions and data in a virus so that critical pieces of data (such as the length of an offset or memory reference locations) are not changed. The reverse engineering module identifies logical blocks of code for the transformation module to manipulate, and the transformation module maps single instructions or blocks of instructions into functionally equivalent code.

 Reverse engineering and transformation techniques are beyond the scope of this chapter. For more details about the structure and functioning of mutation engines and a case study of the Win32.Evol metamorphic virus, see “Are Metamorphic Viruses Really Invincible?” at <http://web.cacs.louisiana.edu/~arun/papers/invincible-complete.pdf>.

Although writing a mutation engine is difficult, the basic function of the mutation engine is simple: introduce useless instructions to fool signature-matching programs. These useless instructions could include:

- Adding zero to a number
- Multiplying by one
- Checking a condition that is always true or always false
- Shifting a bit pattern in one direction and then back in the other direction

Polymorphic viruses still encrypt the virus and the encryption routine, but, in addition, they introduce random useless instructions that scramble the decryption routine. The result is that there is no longer even a small fraction of code for antivirus signature-based detectors to latch onto.

Virus writers now have two techniques for hiding their code: encryption and mutation engines. These can be applied in different ways and result in a few different types of mutating viruses, categorized as

- Oligomorphic virus—A virus that changes its encryptor but not its decryptor
- Polymorphic virus—A virus that changes its encryptor as well as its decryptor
- Metamorphic virus—A virus that changes its encryptor, decryptor, and its *payload*, which is the destructive part of the virus

Metamorphic viruses are especially challenging—even if a decryptor was scrambled, once the virus was decrypted (virus encryption is generally not as difficult to crack as encryption techniques used to keep information confidential), the antivirus program could scan for known fragments of virus code. Metamorphic viruses change the payload of a virus by introducing useless instructions or substituting one set of instructions for another set of functionally equivalent instructions. For example, rather than multiplying a number by four, the code would multiply the number by two twice. Antivirus programs cannot realistically check for all possible combinations of malicious code and useless instructions, so signature-based detections are not sufficient to detect these threats.

## Detecting Complex Viruses

There are a few general techniques for detecting stealth viruses. They are all based on the premise that for a virus to cause damage, it must act on the infected system in some way. Granted, this statement is somewhat like saying that water is wet, but what is important is not so much this first principal as what you can derive from it, such as

- Viruses may change files on the compromised host
- Viruses will make system calls
- Viruses will have patterns of execution different from non-virus programs

Starting with these premises, antivirus designers have developed techniques that complement signature-based detection. Static analysis is the process of analyzing the properties of a program without actually executing it. Compiler designers have used static analysis for generations to optimize compiled code, and similar techniques can be used to detect suspicious patterns of system calls and other indicators of malicious code.

Dynamic analysis techniques execute suspicious code in an emulator and monitor the behavior of the program. Again, suspicious system calls or attempts to intercept system calls can indicate a virus.

Integrity checking techniques examine files for unexpected changes. For example, a checksum is calculated for all executable files and stored for future reference. Periodically, checksums are recalculated and compared with the original checksum. If they are different, the file has changed.

Needless to say, virus writers have responded to these additional detection techniques as well. Responses include:

- Tricking integrity-checking programs to prevent them from detecting changes in checksums by intercepting file I/O and opening uninfected versions of files
- Disguising system calls
- Detecting monitoring programs and changing program behavior in response
- Detecting when the virus is executing within a dynamic analysis emulator and not decrypting code

Again, another iteration of the cat and mouse game is underway.

## State of Virus Detection

Virus writers have circumvented signature-based detection with a variety of stealth techniques and have found ways to avoid dynamic, behavior-based detection. However, antivirus researchers are exploiting the techniques used by polymorphic and metamorphic viruses to detect the malware.

New detection techniques are based on the general methods—rather than the specific instructions—used by viruses to mutate. Mutation engines depend upon program analysis techniques similar to techniques used by compiler writers. Both mutation engines and compilers map one program into a functionally equivalent but structurally different representation. These techniques have theoretical limitations that force them to “show their hand” so to speak.

Exploiting those limitations is not a trivial task and detection techniques are constrained by theoretical limits as well. For example, a classic unsolvable problem in computer science is writing a program that determines whether another program terminates. Known as the Halting Problem, this idea is highly relevant for techniques for analyzing program behavior to determine whether a program is a virus. Virus writers, for example, can include long-running loops in their code knowing that antivirus programs will not generally be able to determine whether a running program will go on forever or eventually stop. Eventually, a behavior analysis program will have to stop a simulated run and decide whether the program is a virus.


### ***Trends in Virus Evolution***

Viruses have evolved along a number of dimensions. They have become progressively more difficult to detect. At first, encryption was used to hide the most obvious identifying patterns of viruses. Next, mutations were introduced to limit the usefulness of signature-based detection. When antivirus designers deployed countermeasures to the radically new mutating viruses, virus writers adopted additional techniques—most notably, attacks on countermeasures, such as hiding system calls, trapping system calls, and manipulating files to hide telltale signs of changes to files.

Both virus and antivirus developers are now working within the theoretical limits of the current static and dynamic analysis techniques. What does the future of virus development look like? It is impossible to say exactly how the continuing cat and mouse scenario will unfold, but the next stages of the virus evolution might realize a number of changes including:

- More sophisticated combination of existing techniques
- More targeted attacks on antivirus programs and other countermeasures
- Exploits of yet-unknown opportunities in new platforms, including 64-bit CPUs and OSs

There are examples of each of these in existence today. Combining different types of malware is common and discussed in detail in the later section on blended threats. Just as antivirus developers study viruses, virus writers as a group have made it a goal to know their enemy and there is no reason to think this trend will end. Viruses targeting 64-bit platforms—such as the proof-of-concept virus, W64/Rugrat—have been discovered. Nonetheless, viruses are not invincible.

 For more information about the W64/Rugrat virus, see [http://vil.nai.com/vil/content/v\\_125990.htm](http://vil.nai.com/vil/content/v_125990.htm).

There are theoretical limits of computation to constrain virus writers. Although it may be challenging, antivirus designers will always have that as a firm foundation to work from, even if those constraints put limits on their work as well. As threatening as they are, viruses are not the only form of malware that must be controlled.

## Worms and Vulnerabilities

Viruses depend upon other programs to function. Like their biological namesake, computer viruses lack some basic machinery to fully function by themselves. Computer worms are like viruses in that they propagate and typically carry malicious payloads; unlike viruses, they do not depend upon other programs to operate.


Worms are fully functional programs that propagate by exploiting vulnerabilities in network software and applications. Two early examples are the Morris Worm and the Melissa worm.

### *Early Worms*

Worms have existed since at least 1978 when Xerox PARC researchers invented the first-known computer worm. The Morris Worm, released in 1988 by a computer science graduate student at Cornell, was the first worm to spread widely on the Internet. The worm took advantage of bugs in three programs distributed with the BSD UNIX OS: sendmail, fingerd, and rsh/rexec.

In 1999, the fastest spreading virus to that date infected email systems around the globe: Melissa. The original version of the virus spread as a macro within Microsoft Word documents that were attached to email. When a recipient of the infected message opened the attachment, the macro virus automatically executed. The virus emailed itself to the first 50 names in the victim's Microsoft Outlook address book. Once activated, the worm would also infect other Word documents on the disk. Variations on the original Melissa worm varied the subject line, emailed 100 instead of 50 contacts, and deleted critical files, including `c:\command.com` and `c:\io.sys`.

Like viruses, worms have evolved from their early days to more sophisticated and threatening forms. Studying the evolution of viruses is instructive for understanding the challenges of detecting malware; similarly, studying examples of worms is instructive for understanding vulnerabilities in systems as well as the emerging incentives for writing malware.

 See Bob Page's "A Report on the Internet Worm" for an analysis of the Morris Worm. The paper is at <http://www.ee.ryerson.ca:8080/~elf/hack/iworm.html>.


### *Implementation Techniques and Consequences*

Worms use a variety of techniques for spreading. Sometimes they email themselves, take advantage of low-level Internet protocols, or exploit a bug in network or application services. The most common transmission methods are

- Email
- Vulnerabilities in Internet protocols or applications, such as IIS
- Instant messaging systems
- Internet Relay Chat (IRC)
- Peer-to-peer file-sharing systems

The consequences of these implementations can vary as well from nuisance, to a Denial of Service (DoS) attack, to a commandeering of computing resources. Some instructive examples of worms are:

- Sobig
- MyDoom
- Sdbot
- SQL Slammer

 For details about these and other worms and viruses, see the McAfee Virus Information Library at <http://vil.nai.com/vil/>.

## Sobig

Sobig is a worm that spreads through email and infects Microsoft Windows OSs. The worm spreads as an attachment to an email message and infects a compromised computer. There were a number of variants of the Sobig worm. After copying itself to a computer, the worm would do the following:

- Copy itself as an .exe file to the Windows system directory
- Create a data file in the Windows system directory
- Add an entry to the Windows registry causing the program to run when Windows starts
- Attempt to copy itself to network shares but failed due to a bug in the program
- Email itself to other victims using a randomly selected email address found on the infected system in the From field of the email.

Variants of Sobig used their own Simple Mail Transfer Protocol (SMTP) program to send themselves to other targets. The payload of the worm was programmed to contact specific Internet addresses and install additional programs including a backdoor that could be used by the worm writer to commandeer the compromised host's resources for spamming or to steal confidential information.

Rather than just leave a Trojan horse program on the infected machine, Sobig attempted to add machines to a collection of network relays that could be used as a distributed computing service. Sobig is illustrative of how worms can spread by email and install Trojans for later use by the worm writer.

## MyDoom

MyDoom, which appeared in January 2004, was another mass mailing worm similar to Sobig in a number of ways, including spreading via email, spoofing email sender information, and setting up back doors for use by the worm programmer. It also expanded beyond the functions found in its predecessor, Sobig, and in the process became the fastest-spreading mass-mailing worm up to that time.



The worm spreads using an attachment that resends itself to email addresses found on the compromised machine. It also copies itself to folders used by peer-to-peer networks, such as KaZaA, allowing it to spread through file-sharing programs as well.

The original version used one payload to establish a backdoor on TCP port 3127 and allow the worm writer to take control of the compromised hosts. A second payload was included to launch DoS attacks against the SCO Group, a UNIX vendor, but did not always function correctly. A later version added a DoS attack against Microsoft. MyDoom is an example of a fast-spreading worm that uses two vectors of transmission, email and peer-to-peer file sharing.

## Sdbot

The Sdbot worm appeared in February 2005 and spreads through instant messaging and takes advantage of a buffer overflow vulnerability in the Windows remote procedure call (RPC) interface that allows the attacker to run programs on the compromised host. It also uses a vulnerability in the Local Security Authority Subsystem Service (LSASS) to execute arbitrary code. Once on a machine, the worm copies itself in several ways:

- Insecure network shares
- Poorly secured SQL Server database instances
- Poorly secured MySQL database instances

Once on a machine, the worm installs itself in the Windows system directory and changes the system registry to run the program on startup. The payload of this virus includes a *bot*, or software agent, that can download and execute programs as directed by the worm's writer or other controller. McAfee identified a number of remote access functions that can be performed by the worm, including:

- View, modify registry data
- Browse filesystem
- Browse, terminate, start processes
- Upload, download, delete, modify, execute files
- Run FTP server
- Run HTTP proxy
- Run SOCKS proxy
- Manipulate (add, remove, modify) shares on victim
- Log keystrokes
- Launch DoS attack from victim machine
- Perform network scans (locate other vulnerable machines)

With these functions, the attacker can effectively take control of a machine and use it for a number of illegitimate uses, including stealing personal information and sending spam. One of the most dangerous types of programs that an attacker can download is a rootkit. Rootkits are sets of tools that allow an attacker to control a computer while disguising its actions. Rootkits are extremely difficult to detect; the best solution is to prevent them from reaching computers in the first place.

Sdbot is illustrative of the use of IRC and instant messaging systems to propagate. Like other worms, Sdbot leaves backdoor programs and Trojans, creating the opportunity for further exploitation by the attacker.

## SQL Slammer

SQL Slammer is probably one of the best known Internet worms. In January 2003, a small 376-byte program effectively shutdown large segments of the Internet by flooding the network with useless traffic. The damage caused by some worms, such as Sdbot and MyDoom, is caused by their relative sophistication. In the case of SQL Slammer, the damage was due to several factors, including:

- The ability to exploit a single, widespread vulnerability in Microsoft SQL Server
- The speed of the protocol used, which was the User Datagram Protocol (UDP)
- The inherent trust (lack of authentication) in the protocol that was exploited

### ***Database Vulnerability***

The worm took advantage of two vulnerabilities in the SQL Server Resolution Service, a service designed to ensure database requests are sent to the correct instance of SQL Server when multiple instances are running on a single machine. Like other buffer overflow exploits, this one takes advantage of the fact that the SQL Server Resolution Service expects a piece of data to be at most a certain length (in this case, a 16-byte database name). The worm author crafted a packet of information with data longer than 16-bytes, causing a buffer overflow. (Actually, there were two different overflows, one overwrote the *heap*, a data storage area; the other overwrote the *stack*, the instruction storage area.)

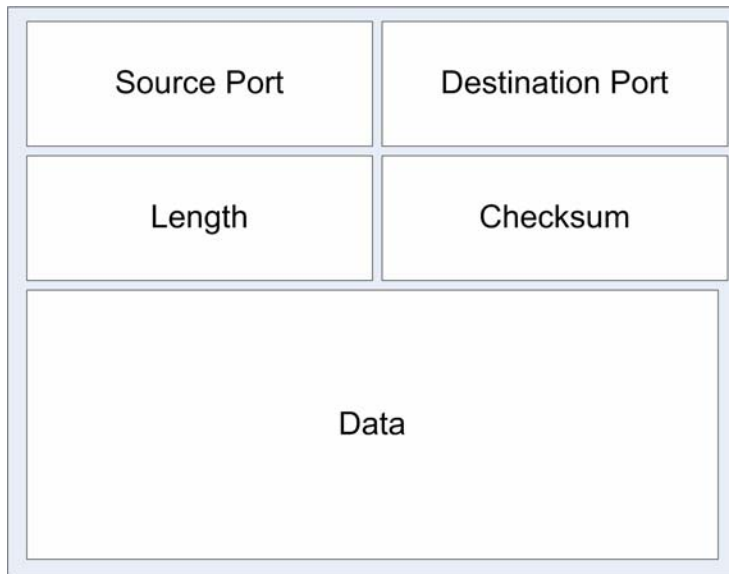
The SQL Server Resolution Service did not detect an error but simply continued to execute; however, now it was executing code written by an attacker instead of Microsoft developers. The program was simple:

- Determine the number of milliseconds since the server was booted
- Use the number of milliseconds as a destination IP address in a UDP packet
- Copy the program to the data segment of the UDP packet
- Repeat

Of course, randomly generating IP addresses will generate some that do not resolve to actual servers and many that are not running SQL Server. However, the speed at which the program was able to execute and speed of UDP made this simple but rapid attack quite effective.

**UDP—The Minimalist Protocol**

UDP is as simple a network protocol as one will find. Figure 3.5 shows the structure of a UDP Packet.



**Figure 3.5:** The UDP packet contains only address information, basic data integrity data, and the data itself.

UDP is one of the two network-level protocols used by the Internet. The other, TCP, is much more complex. When data must be reliably delivered from source to destination in the correct order and with some control over the speed at which packets are sent, then TCP is used. This protocol provides more guarantees about the information delivered but at the cost of significantly higher overhead.


UDP is used when packets can be sent but acknowledgement is not required. Packets can arrive out of order without causing problems for the recipient. In fact, packets may not be delivered at all, but services that use this protocol have a higher tolerance for failure than those that use TCP. Programs that use UDP are typically network services, such as:

- Domain Names Service (DNS)
- Dynamic Host Configuration Protocol (DHCP)
- Simple Network Management Protocol (SNMP)
- SQL Server Resolution Service
- Routing Information Protocol (RIP)

When programs use these services, they exchange small amounts of information and, if necessary, can request again and resend information without adversely affecting the source and target applications. The simplicity of UDP is also its weakness. The protocol has no flow control mechanism, so a network can be flooded with UDP packets to the point that other services cannot function. This is what happened with SQL Slammer.

### Trusting Protocols

Another factor that affects both TCP and UDP is that they inherently trust the source of a packet. There is no authentication of a server with these network-level protocols (there is with other protocols, such as SSL/TSL). In the case of UDP, there is no mechanism to slow or discard packets that are clearly flooding the network. Furthermore, it is not at all clear that these services should be at the network protocol layer. High volumes of traffic from a single source are better controlled by routers and intrusion prevention systems than at the lower levels of the network.


 For a detailed description of the impact of the SQL Slammer worm, see Paul Boutin's "Slammed: An Inside View of the Worm that Crashed the Internet in 15 Minutes" at <http://www.wired.com/wired/archive/11.07/slammer.html>.

### Unseen Vulnerabilities—Lack of Knowledge

The impact of SQL Slammer did not have to happen. Microsoft published a patch for the buffer overflow vulnerabilities in the SQL Server Resolution Service 6 months before the attack. Obviously, there were tens of thousands of unpatched servers in on the Internet. Some systems administrators may not have known about the patch and others may have known about it and choose not to apply it.

This reaction might sound irresponsible to some, but as any seasoned systems administrator knows, applying a patch can break as well as fix. Organizations that depend on SQL Server, or any production application for that fact, for essential services should not apply patches without testing them in a controlled environment. In addition to ignorance about the existence of the vulnerability and the patch, there are some users that probably did not know they were running a version of SQL Server known as Microsoft SQL Server Desktop Engine (MSDE), which is licensed free of charge with some applications.

SQL Slammer is illustrative of the fact that worms need not be complex to cause significant damage. It also illustrates the existence of inherent vulnerabilities in basic Internet protocols as well as enterprise applications.

 Databases should never be placed unprotected on the Internet. Database servers should at least be behind a firewall. A better option is to make the database server accessible only to an application server that is also protected behind a firewall.

### Increasing Malevolence

Worms are becoming more destructive. Early worms, such as the Morris Worm, disrupted Internet services, but systems administrators were able to recover without long-term damage. Worms such as Sdbot and SQL Slammer show that malware developers are effectively exploiting vulnerabilities throughout the Internet infrastructure and in application services. Like viruses, worms are becoming more destructive. As if that were not enough to keep antivirus developers on their toes, malware developers have combined techniques to form a new type of malware—the blended threat.

## The New Frontier—Blended Threats

Blended threats are malware that use multiple techniques to spread and cause damage. A blended threat may contain several of the following:

- Virus
- Worm that propagates by multiple methods
- Trojan horses
- Keylogger
- Frame grabber
- SMTP email engine
- Rootkits
- Backdoors

Any one of these types of malware can cause significant damage; in combination, the potential is much greater. Blended threats were launched as early as 2001 with the release of the Code Red and Nimda worms. Code Red launched Distributed DoS (DDoS) attacks, changed registry settings, created network shares, and spread via email and vulnerable IIS Web servers. Nimda had similar characteristics and even went so far as to attack backdoors left by Code Red II. Some of the most notable characteristics of blended threats are their

- Methods of attack
- Methods of transmission
- Methods of control

### ***Multiple Methods of Attack***

Blended threats use multiple methods of attack. This behavior increases the chance of success (that is, from the malware developers point of view) when deployed into environments with countermeasures in place. For example, laptops with up-to-date antivirus software may not be vulnerable to a virus infection from a polymorphic virus but it could be vulnerable to a worm that exploits a vulnerability in the Microsoft SQL Server Desktop Engine.

Attacks can come in the form of:

- Deleting files
- Changing registry settings
- Infecting executable files
- Deploying backdoors for later use
- Disabling antivirus software

Although vandalism malware is still a threat (for example, defacing a Web site), stealing information and computing resources are the real objectives of these attacks. Identity theft is a growing problem because identities have replaced cash as the object of criminal desire. Times have changed from when famed bank robber Willie Sutton targeted banks because “that’s where the money is.” The location and form of the money has changed; it is now in computers in the form of personally identifiable information.

In addition to personal information, just having access to the bandwidth and computing cycles of a compromised machine has economic value. Anti-spam legislation makes it difficult to carry out spamming activities in the open with fixed servers and known IP addresses. Instead, spammers turn to attackers who have commandeered collections of PCs through the use of backdoors and rootkits. Need to deliver a million pieces of spam? Find the right chat room, and you can rent a set of compromised machines to do the mailing for you. Attacks are changing and they are reflecting changes in the underlying motives of the attackers. Economics are driving malware.

### ***Multiple Methods of Transmission***

Blended threats may include virus-like components, but they propagate without the use of other programs; in that way, they are a form of a worm. The Klez family of worms, for example, spreads by taking advantage of a bug in Microsoft Outlook’s Automatic Execution of Embedded MIME Type, which does not require any action on the part of the user. The malware then spreads using its own email engine after collecting email addresses from the compromised host. It also propagates by copying itself to network shares.

Typical transmission methods include:

- Employing virus-like file infections
- Copying to network shares
- Copying to peer-to-peer network shares
- Exploiting vulnerabilities in Web servers or other system applications
- Using email


Once they have transmitted and attacked, blended threats do not necessarily shutdown or cease their malicious activity.

### ***Multiple Methods of Control***

As noted earlier, the motives for hacking and attacking are changing. Economics is an established incentive in the world of malware. One of the ways cybercriminals can realize a return on their investment in malware development is by gaining control of a large number of computers.

Zombies, or computers that are available for control by attackers, are essentially free resources for malware developers. Once a worm has reached a PC and installed a backdoor program, the program can download updates, install additional malware, or even launch a mass mailing of spam. Computationally intensive tasks, such as cracking encryption keys, could also be parceled out to zombies.

Blended threats combine the techniques learned over decades of development and evolution of viruses and worms and now apply them to ever-more damaging activities. The world of attacking is changing; it is no longer the stereotypical lone hacker breaking into a system to demonstrate his or her technical prowess. Organized crime has discovered that money is not just in banks anymore.

 See “Hacker Hunters: An Elite Task Force Takes on the Dark Side of Computing” for an example of how one cybercrime group, ShadowCrew, was broken by the United States Secret Service and FBI. The story is available at [http://www.businessweek.com/magazine/content/05\\_22/b3935001\\_mz001.htm](http://www.businessweek.com/magazine/content/05_22/b3935001_mz001.htm).

## Summary

Viruses began as the intellectual brainchild of computer science researchers in the 1960s and were rediscovered by teenagers with early PCs. Worms appeared later but followed a similar trajectory of evolution. Today, we’re confronted with a host of well-known techniques for creating, deploying, hiding, and modifying malicious programs. We are also becoming painfully aware of the vulnerabilities that exist in the trusted code that runs our organizations and serve us in home use. The combination of the two is challenging to say the least.

Fortunately, there are countermeasures. Antivirus and anti-worm software has co-evolved along with threats. Improved security practices and system patching procedures can minimize vulnerabilities. Controlling malware is not a project that can be completed; we will always have to defend against it.

## Content Central

[Content Central](#) is your complete source for IT learning. Whether you need the most current information for managing your Windows enterprise, implementing security measures on your network, learning about new development tools for Windows and Linux, or deploying new enterprise software solutions, [Content Central](#) offers the latest instruction on the topics that are most important to the IT professional. Browse our extensive collection of eBooks and video guides and start building your own personal IT library today!

## Download Additional eBooks!

If you found this eBook to be informative, then please visit Content Central and download other eBooks on this topic. If you are not already a registered user of Content Central, please take a moment to register in order to gain free access to other great IT eBooks and video guides. Please visit: <http://www.realtimepublishers.com/contentcentral/>.