Computer Virus–Coevolution

The battle to conquer computer viruses is far from won, but new and improved antidotes are controlling the field.

Carey Nachenberg

As recently as six years ago, computer viruses were considered an urban myth by many. At the time, only a handful of PC viruses had been written and infection was relatively uncommon. Today the situation is very different. As of November 1996, virus writers have programmed more than 10,000 DOS-based computer viruses.

In addition to the sheer increase in the number of viruses, the virus writers have also become more clever. Their newer creations are significantly more complex and difficult to detect and remove. These "improvements" can be at least partially attributed to the efforts of antivirus producers. As antivirus products improve and detect the "latest and greatest" viruses, the virus authors invent new and more devious ways to hide their progeny.

This coevolution has led to the creation of the most complex class of virus to date: the *polymorphic* computer virus. The polymorphic virus avoids detection by mutating itself each time it infects a new program; each mutated infection is capable of performing the same tasks as its parent, yet it may look entirely different.

These cunning viruses simply cannot be detected cost-effectively using traditional antivirus scanning algorithms. Fortunately, the antivirus producers have responded, as they have in the past, with an equally creative solution to the polymorphic virus threat. Many antivirus programs are now starting to employ a technique known as *generic decryption* to detect even the most complex polymorphic viruses quickly and cost effectively.

A computer virus is a self-replicating computer pro-

BEATA SZPURA

—Antivirus

gram that spreads by attaching itself to executable files or system areas on diskettes. Recently, we have also encountered a new type of virus that infects application data files that contain macros. These viruses are constructed entirely of application macros and use the macro language to propagate themselves.

In addition to their ability to replicate, some computer viruses also deliver a *payload*—a portion of the virus program that is designed to damage the host machine, display a message, or do some other mischief without the computer operator's consent. This article focuses primarily on how computer viruses replicate and obscure themselves.

The vast majority of computer viruses have been designed specifically for IBM-based PCs running the DOS and Windows operating systems. In terms of sophistication and functionality, these DOS-based viruses are generations ahead of viruses written for other operating systems and platforms. Consequently, this article examines how the antivirus community has tackled these DOS-based viruses. Nonetheless, the concepts presented apply for viruses found on all operating systems and computer platforms.

The first file-infecting viruses were simple machine language programs that had the ability to attach and spread identical copies of themselves from program to program. When the user executed an infected program, the virus would take control of the computer and infect additional files. After the virus completed its mischief, it would transfer control to the host program and allow it to function normally. This type of virus is called a "parasitic" computer virus, since it does not kill its host; instead, the host acts as a carrier.

Since these viruses replicated identical copies of their machine code and data each time they infected a new program, they proved easy for antivirus products to detect. Every time the antivirus researcher, received a new virus, they would analyze the virus and identify a sequence of machine-language instructions that were both unique to the virus and present in every one of its infections; this sequence of bytes is known as a virus signature. The researcher could then insert this signature into the antivirus program so that it could search for the virus.

The first antivirus programs were glorified string-searching programs. The antivirus program would open each executable file on the computer and scan its entire contents for a current set of virus signatures. If any of the signatures were found anywhere in the target program, the antivirus program would report the infection to the user. If none of the signatures were found, the antivirus program would report that the file was uninfected.

This detection technique worked well for quite a while. However, as the number of viruses steadily increased, antivirus programs had to become more clever in order to run at reasonable speeds. After all, scanning for hundreds of virus signatures through hundreds of kilobytes of executable programs was a very slow process, especially on 4.77 megahertz computers! These speed problems led to one of the great innovations in early antivirus programs.

MOST COMPUTER VIRUSES, INCLUDING MANY of today's complex viruses, either append themselves onto the end or place themselves at the beginning of their host executable file. Furthermore, almost all computer viruses are less than 4KB in length. Antivirus researchers recognized these patterns and optimized their virus scanners accordingly. The result of this improvement was antivirus programs that could detect the vast majority of file viruses by scanning less than 8KB of each executable program. This technique dramatically increased the speed and efficiency of antivirus products and is still being used today in some products.

While these improvements significantly improved the speed of the virus scanner, even more optimizations were possible. For example, almost all file-infecting viruses infect at the entry-point of their host program. The entry-point of a program is the location in the program of the first machine-language instruction that is executed when the program is launched.

As an analogy, consider a notepad with a list of instructions on how to complete a certain task, such as sending a fax. (See Diagram 1A.) In order to send a fax, a person can simply follow the instructions on the pad from start to finish. Computers interpret programs, which are basically sequences of simple instructions, in much the same way. In our notepad example, the entry-point of the notepad is the line that contains the first instruction in the list. This would be the first instruction that a human would look at when they start on their task.

When a virus infects a new program, it places itself at the entry-point of the host program or modifies the machine-language instructions at the host's entry-point to transfer control to its virus body. In the latter case, the virus body is usually located at the end of the host program. (See Diagram 1B.)

When the user executes a program, the operating system loads the program into the computer's RAM and then instructs the CPU to start executing the instructions at the program's entry-point. Viruses infect at the entry-point of

their host because it guarantees that when the user executes a program, the virus is immediately given control of the computer.

Since most computer viruses were found to infect or modify the entry-point of a host program, antivirus authors realized they could make their AV programs even faster. They redesigned their scanners to scrutinize only the instructions that can be reached directly from the entry-point in each executable program. This type of honed search is known as entry-point scanning. The entry-point scanner works in the following manner:

1. Establish a variable E contains the target program's entry-point location.
2. The entry-point scanner examines the machine-language instruction at location E in the suspected program.
3. If this instruction transfers control to another instruction (as in Diagram 1B), then set E to the location of the destination instruction and go back to step 2.
4. Search the bytes at location E for virus signatures.

Therefore, if this algorithm were applied to the program in Diagram 1A, the entry-point scanner would finish with an E value of 1, and search for viruses starting at the very first instruction. If applied to the program in Diagram 1B, the entry-point scanner would finish with an E value of 6, and search starting for viruses at the first viral instruction.

Clearly this technique greatly reduces the number of bytes that must be searched for viruses and dramatically improves antivirus performance. This entry-point scanning technique continues to be used by many popular antivirus programs.

Unfortunately, these early advances were soon countered by the virus writers who realized they could make their viruses more difficult to detect if the viruses did not spread exact copies of themselves from file to file. This led to the creation of the *encrypted* virus.

The encrypted virus consists of two parts: a small program known as the virus decryption routine, and an encrypted virus body. The virus decryption routine is composed of a sequence of machine-language instructions that can decrypt the encrypted virus body. The encrypted virus body is just an encrypted version of the same virus machine-language instructions used by the simple viruses described earlier.

If a user executes an infected program, the virus decryption routine immediately executes and decrypts the machine-language instructions that comprise the rest of the virus. The decryption routine then transfers control to the newly decrypted virus body and allows the virus to execute normally. As you can see, the virus body is visible only when the virus is executing and in memory.

When the virus infects a new program, it re-encrypts a copy of the virus body (using a complementary encryption program that is also carried along in the virus body) and appends this onto the host program. It also appends the decryption routine onto the host program. Most encrypting viruses use a different encryption key each time they infect a new program; consequently, the encrypted virus body appears different in each infection. This means the virus decryption program constitutes the only consistent, visible sequence of instructions that are passed from infection to infection. (See Diagram 2.)

The encrypted virus posed new problems for antivirus researchers. In the past, we could detect viruses by searching for virus signatures extracted from the virus body. Clearly, this is not an option with the encrypted virus since it doesn't maintain a consistent virus body from infection to infection. However, the encrypted virus does propagate an identical copy of its small decryption routine from infection to infection. Could this be enough to detect the virus?

As it turns out, the answer is "yes" in most cases. The decryption routines used by most encrypting viruses are often unique. Most of them are made up of at least 10 to 15 distinct bytes. In fact, many antivirus researchers can look at a disassembly of a virus decryption routine and immediately identify the virus' strain without decrypting and examining the rest of the virus!

Because of this, we were able to use the same virus signature scanning technique to detect many of the encrypting viruses. However, there was one drawback: most virus

Before **A.**

1	Insert document in fax machine. (Program entry-point)
2	Dial the phone number.
3	Hit the SEND button on the fax.
4	Wait for completion. If a problem occurs, go back to step 1.
5	End task.
6	
7	
8	
9	

After **B.**

1	Skip to step 6. (Virus modified entry-point, transfers control to the virus body on line 6.)
2	Dial the phone number.
3	Hit the SEND button on the fax.
4	Wait for completion. If a problem occurs, go back to step 1.
5	End task.
6	VIRUS instructions
7	VIRUS instructions
8	VIRUS instructions
9	Insert document in fax machine. (Stored by the virus)

Diagram 1. Notepad example before and after infection at the entry-point

decryption routines were compact machine-language programs. Consequently, our virus detection signatures for these viruses were also short, and as signatures got smaller, the likelihood of false identification and misidentification increased.

False identification occurs when an antivirus program incorrectly identifies an uninfected program as being infected. Misidentification occurs when an antivirus program correctly identifies that a program is infected, yet improperly identifies the strain of the infection. Both of these results are undesirable. If an antivirus product mistakenly identifies a program as being infected, significant time and money must be spent to determine whether or not the infection is legitimate.

In order to solve these problems, we made one additional improvement to our virus scanner. We modified it so that it could perform a secondary verification when it located a questionable virus decryption routine signature. The scanner would attempt to decrypt the contents of the would-be virus with the assumption the virus employed one of a number of simple encryption techniques. As it turns out, a high percentage of encrypted viruses use easily decryptable encryption schemes, so this verification was often successful.

Encrypted with a key value of 1 (see line 6). Notice how lines 7, 8 and 9 are encrypted; compare with Figure 1B. **A.**

1	Skip to step 6.
2	Dial the phone number.
3	Hit the SEND button on the fax.
4	Wait for completion. If a problem occurs, go back to step 1.
5	End task.
6	Starting at line 7, shift back each letter by one. B goes to A, T goes to S, etc. (Virus decryption loop)
7	WJSVT jotusvdujnot (Encrypted "VIRUS instructions")
8	WJSVT jotusvdujnot (Encrypted "XIRUS instructions")
9	Jotfsu epdvnfou jo gby nbdijof. (Encrypted "Insert document in fax machine.")

Encrypted with a key value of 2 (see line 6). **B.**

1	Skip to step 6.
2	Dial the phone number.
3	Hit the SEND button on the fax.
4	Wait for completion. If a problem occurs, go back to step 1.
5	End task.
6	Starting at line 7, shift back each letter back by two. C goes to A, U goes to S, etc. (Virus decryption loop)
7	XKTWU kpuvtwevkopu (Encrypted "VIRUS instructions")
8	XKTWU kpuvtwevkopu (Encrypted "VIRUS instructions")
9	Kpugtv fqewogpv kp hcz ocejkpg. (Encrypted "Insert document in fax machine.")

Diagram 2. The encryption virus, two different infections

By performing this secondary verification, the antivirus program could definitively identify whether or not the program in question was infected by a virus. This technique has been named "x-raying," since the antivirus program looks through the encryption at the insides of the virus.

These encrypted viruses pose few problems for modern antivirus programs; they can be detected quickly and easily. Today, however, the polymorphic virus provides huge headaches for antivirus researchers.

The Polymorphic Virus

With biological viruses, mutations overwhelmingly result in nonviable offspring. However, because of their sheer numbers, at least some of the mutated offspring are successful. Computer viruses cannot afford to play this type of Russian roulette.

If a simple computer virus were to propagate randomly mutated copies of itself, the odds are the mutated children would fail to exhibit virus-like properties. In the most likely scenario, a mutated child virus would cause its host program to crash any time it was executed. This would immediately reveal the virus to the user and limit its ability to successfully reproduce.

Because of these realities, current polymorphic computer viruses do not mutate at all; instead, they have specially designed *mutation engines* that simulate the process of mutation.

The polymorphic virus is basically an encrypted virus with a twist. Recall the simple encrypted virus carries an unchanging machine language decryption routine from host to host. An antivirus product can be programmed to search for the static bytes that comprise this decryption routine.

To address this problem, the polymorphic virus uses its mutation engine to generate a new decryption routine each time it infects a new program. The newly constructed decryption routine is the same functionally as those found in other infections. However, the sequence of instructions that comprise this routine may be entirely different.

The mutation engine also generates a complementary encryption routine that is used to encrypt the static portion of the virus before it is attached to a new target file. Once a copy of the virus body has been encrypted (using this complementary encryption routine), the virus appends the newly generated decryption routine along with the encrypted virus body onto the target executable file. Thus, not only is the virus body encrypted, but the virus decryption routine uses a different sequence of machine-language instructions in each infected program. Since the virus decryption routine varies from infection to infection and the rest of the virus is encrypted, antivirus programs can not detect the virus by searching for fixed signatures.

The virus mutation engine is a complex program. In fact, most mutation engines are far more complex than

their accompanying virus. The mutation engine must be able to produce seemingly random programs that can properly perform decryption. Some of the more complex mutation engines can generate billions of billions of billions of different decryption routines, making simple signature detection impossible.

While many polymorphic viruses can be detected using augmented versions of the entry-point and head-tail scanning algorithms, until recently, antivirus researchers had no efficient way to detect the more complex strains. In the past, researchers wrote specialized detection programs in C, assembly language, or special virus detection script languages to catch the these viruses. These detection programs examined the machine code at the entry-point of each file. If they found sequences of instructions resembled those generated by a given mutation engine, they would report that the file was infected.

Many antivirus companies still write such specialized detection programs today. In fact, with some of the more complex viruses, it is not feasible to detect the virus using any other technique. Unfortunately, the creation of these hand-coded, virus-specific detection programs requires thorough virus analysis and significant programming effort by a trained antivirus researcher. In addition, because these detection programs work based on different principles than the underlying scanner, they often significantly reduce the efficiency of the antivirus product.

Worst of all, these programs are predisposed to false identification and misidentification. The decryption routines created by the more complex engines can take so many different forms that it is often difficult to discern between a virus decryptor and a legitimate program.

The Solution

Over the past two years, several antivirus companies have incorporated generic decryption (GD) technology into their virus scanners. This technology enables the antivirus program to detect even the most complex polymorphic viruses with ease, while maintaining fast scanning speeds. Using GD, an antivirus researcher can update a virus scanner to detect new polymorphic viruses within minutes or hours instead of days or months.

Recall that all current polymorphic viruses have a body of machine code that is encrypted and copied verbatim from infection to infection. When a program infected with a polymorphic virus is launched by the user, the polymorphic decryption routine executes and decrypts this unchanging virus body. Once the decryption routine finishes decrypting the body, it transfers control to the virus so it can spread. Thus, when a file containing a polymorphic virus is executed, the virus is guaranteed to decrypt itself and reveal its innards; otherwise, it would not be able to execute and constitute a viral threat. The GD technique relies on this behavior in order to detect polymorphic viruses.

The GD scanner is comprised of a CPU emulator, a virus signature scanner, and an emulation control module (ECM). When the user requests a scan of an executable file, the GD antivirus loads the suspect program into a software-simulated, virtual computer. The program is then allowed to execute in this virtual computer as if it were running on a real computer. During this execution, if the target file is infected with a virus, it can cause absolutely no damage to the actual PC since it is executing in a completely contained environment.

At the start of the simulation, the emulator begins executing the instructions at the infected program's entry-point. These instructions usually transfer control to, or directly constitute, the virus' decryption routine. Just as on a real computer, the decryption routine wastes no time deciphering the body of the virus. The emulation control module regulates the emulation and continues it as long as its required.

The GD constantly monitors the progress of the emulation session. At regular intervals, it calls upon its signature scanner to search through modified and potentially decrypted regions of virtual memory for virus signatures. Effectively, the virus does all the decryption work for the antivirus program. As a result, even viruses that employ arbitrarily complicated encryption schemes can be detected with ease. Furthermore, the GD scanner can exactly identify the strain of the virus since it has access to the entire, decrypted virus body.

In essence, this is like injecting a mouse with a serum which may or may not contain a virus, and then observing the mouse for any adverse effects. If the mouse becomes ill, (i.e., the virus manifests itself) we can observe the visible symptoms and identify the virus. If the mouse stays healthy, then we can select another vial of serum and repeat the process.

The benefit of GD is it provides accurate identification of polymorphic viruses and dramatically reduces the possibility of false identification or misidentification. This behavior results because the scanner detects viruses by examining the static virus body instead of the polymorphic section of the virus, which can take numerous forms.

Unfortunately, it is not such an easy chore to create the perfect generic decryption scanner. Perhaps the most difficult task in constructing a GD scanner is designing the emulation control module. If the GD scanner knew that it would always be called upon to scan infected files, it would be trivial to design a perfect ECM. Its one and only rule would be: emulate the target program until one of the viral strains is definitively identified, then quit.

This situation would be ideal but it is not realistic. The majority of users do not have virus-infected files on their computer. Therefore, the ECM must decide how long to

emulate each program before giving up in its search for viruses. For example, if the ECM were to emulate each program for five minutes, it would allow the GD scanner to catch almost all polymorphic viruses. However, the antivirus program would be too slow for the average user.

Conversely, if the ECM only emulated the first 15 instructions of each program, unless it observed some decryptor-like behavior, it might miss some viruses. For instance, some polymorphic viruses employ "do-nothing" instructions within the polymorphic decryption routine to conceal their presence. If the ECM does not emulate enough instructions, it may be fooled by these do-nothing instructions and terminate emulation prematurely.

The difficulties in designing an ECM may remind some readers of the famous Turing Halting Problem. The great mathematician Alan Turing proved it is impossible to construct a program that can always determine, in finite time, whether or not another arbitrary program will ever terminate (or whether it will go on running for ever). The ECM must perform an equally daunting task; it must decide whether or not the current program could decrypt itself to unleash a virus.

The encrypted virus does propagate an identical copy of its small decryption routine from infection to infection. Could this be enough to detect the virus?

There is currently no consensus among antivirus researchers on how to design an efficient ECM, and each antivirus product has its own unique implementation.

Is GD The Solution?

GD has proved to be the most cost-effective method for detecting the widest variety of polymorphic viruses to date. However, while GD performs exceptionally well, it is no panacea. There exist entire classes of polymorphic viruses that cannot be detected using the GD technique. Even more worrisome is the fact these detection-resistant viruses are potentially no more difficult to program than current polymorphic viruses; they are merely different.

Consider the following GD problem: The CPU emulator used in a GD implementation may be designed to simulate a single CPU architecture, such as an 80486. While the 80486 chip is largely compatible with earlier CPUs, it does have some significant differences at the machine-language level.

If a virus writer were to design a virus decryption routine that used 80286-specific machine-language instructions, then the 80486 CPU emulator might be unable to properly emulate and decrypt the virus. Granted, the virus would be unable to function on a real 80486-based computer either. However, it still might be a viable virus on many older machines.

How might a GD antivirus program handle this problem? Well, the obvious answer is that it could be designed to simulate every type of CPU for all scanned files. Each and every scanned program would be emulated 20 different times, for each of the different major types of processors and in-line processor revisions. Unfortunately, this would dramatically slow down the antivirus product to the point where it would be unusable.

Alternatively, the virus could be detected using a more traditional technique, such as a hand-coded antivirus module. This solution is practical if it is required for only a few viruses. However, it becomes unmanageable as the number of viruses of this type becomes large.

Conclusions

Computer viruses have dramatically increased in complexity over the years. The first viruses were static programs that copied themselves from program to program or diskette to diskette. The viruses that exist today are much more complex. Many utilize convoluted, multilevel encryption schemes and do not have any visible, consistent machine-language instructions from infection to infection. These elaborate viruses have placed a strain on antivirus producers.

With the use of GD, researchers can make their antivirus programs detect the majority of current polymorphic viruses without expending significant additional effort above that required for normal viruses. This is a huge step in the antivirus community. In the past, many antivirus companies simply could not afford to allocate scarce engineering resources to analyze the small minority of polymorphic viruses.

While we are far from winning the virus-antivirus war, the current polymorphic virus battle is under control. It is an extremely difficult task to construct a complex polymorphic virus; virus writers often spend weeks or even months to create an elaborate mutation engine. With GD technology, these same viruses can be detected in just minutes or hours by an antivirus researcher. This improved efficiency means we can spend more time detecting the scores of normal viruses, researching new antivirus technologies, and protecting the user. ◻

Carey Nachenberg *(cnachenberg@symantec.com) is a principal software engineer at the Symantec Antivirus Research Center, Symantec Corp., Santa Monica, Calif.*