Understanding and Managing Polymorphic Viruses

Understanding and Managing Polymorphic Viruses
Carey Nachenberg
[Symantec 1996]

Introduction

Polymorphic computer viruses are the most complex and difficult viruses to detect, often requiring anti-virus companies to spend days or months creating the detection routines needed to catch a single polymorphic.

This white paper provides an overview of polymorphics and existing methods of detection, and introduces Symantec's Striker ™ technology, a new, patent-pending method for detecting polymorphics.

Norton AntiVirus 2.0 for Windows 95 is the first Symantec anti-virus product to include Striker; Symantec will integrate Striker into other Norton anti-virus products as it introduces new editions.

The Evolution of Polymorphic Viruses

A computer virus is a self-replicating computer program that operates without the consent of the user. It spreads by attaching a copy of itself to some part of a program file, such as a spreadsheet or word processor. Viruses also attack boot records and master boot records, which contain the information a computer uses to start up. Macro viruses attack such files as word processing documents or spreadsheets.

Most viruses simply replicate. Some display messages. Some, however, deliver a payload — a portion of the virus program that is designed to corrupt programs, delete files, reformat a hard disk, or crash a corporate-wide network, potentially wiping out years of data and destroying critical information.

Simple Viruses

A simple virus that merely replicates itself is the easiest to detect. If a user launches an infected program, the virus gains control of the computer and attaches a copy of itself to another program file. After it spreads, the virus transfers control back to the host program, which functions normally.

Yet no matter how many times a simple virus infects a new file or floppy disk, for example, the infection always makes an exact copy of itself. Anti-virus software need only search, or scan, for a tell-tale sequence of bytes — known as a signature — found in the virus.

Encrypted Viruses

In response, virus authors began encrypting viruses. The idea was to hide the fixed signature by scrambling the virus, making it unrecognizable to a virus scanner.

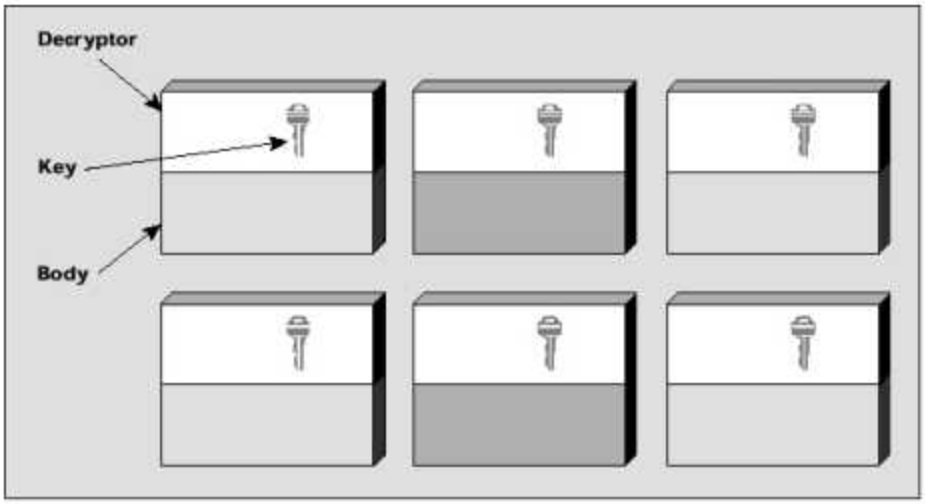

Figure 1. An encrypting virus always propagates using the same decryption routine. However, the key value within the decryption routine changes from infection to infection. Consequently, the encrypted body of the virus also varies, depending on the key value.

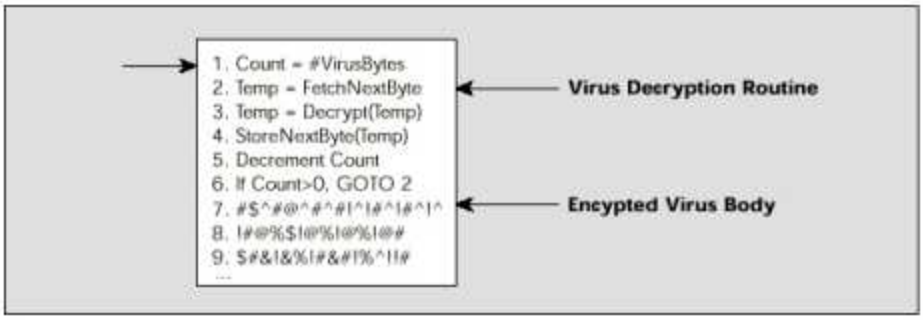

Figure 2. This is what an encrypted virus looks like before execution.

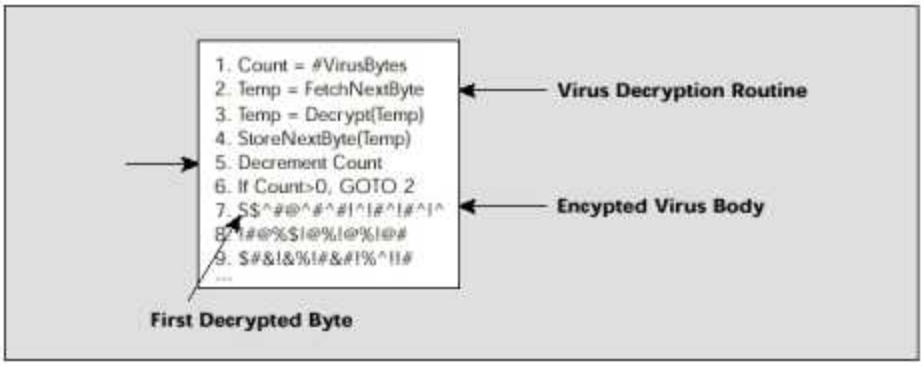

Figure 3. At this point, the virus has executed its first five instructions and has decrypted the first byte of the encrypted virus body.

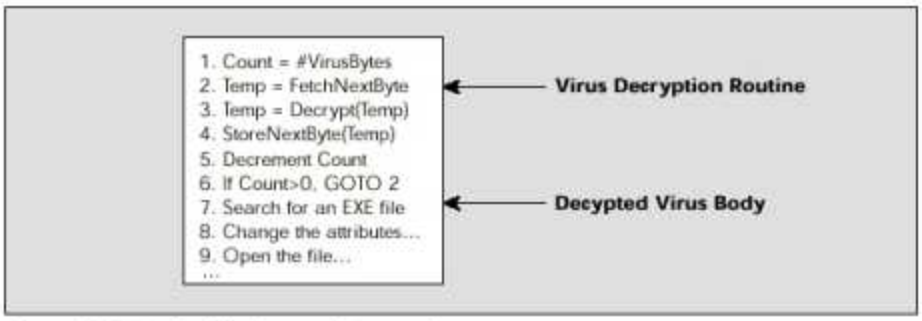

Figure 4. This is the fully decrypted virus code.

An encrypted virus consists of a virus decryption routine and an encrypted virus body. If a user launches an infected program, the virus decryption routine first gains control of the computer, then decrypts the virus body. Next, the decryption routine transfers control of the computer to the decrypted virus.

An encrypted virus infects programs and files as any simple virus does. Each time it infects a new program, the virus makes a copy of both the decrypted virus body and its related decryption routine, encrypts the copy, and attaches both to a target.

To encrypt the copy of the virus body, an encrypted virus uses an encryption key that the virus is programmed to change from infection to infection. As this key changes, the scrambling of the virus body changes, making the virus appear different from infection to infection. This makes it extremely difficult for anti-virus software to search for a virus signature extracted from a consistent virus body.

However, the decryption routines remain constant from generation to generation — a weakness that anti-virus software quickly evolved to exploit. Instead of scanning just for virus signatures, virus scanners were modified to also search for the tell-tale sequence of bytes that identified a specific decryption routine.

Polymorphic Viruses

In retaliation, virus authors developed the polymorphic virus. Like an encrypted virus, a polymorphic virus includes a scrambled virus body and a decryption routine that first gains control of the computer, then decrypts the virus body.

However, a polymorphic virus adds to these two components a third — a mutation engine that generates randomized decryption routines that change each time a virus infects a new program.

In a polymorphic virus, the mutation engine and virus body are both encrypted. When a user runs a program infected with a polymorphic virus, the decryption routine first gains control of the computer, then decrypts both the virus body and the mutation engine. Next, the

decryption routine transfers control of the computer to the virus, which locates a new program to infect.

At this point, the virus makes a copy of both itself and the mutation engine in random access memory (RAM). The virus then invokes the mutation engine, which randomly generates a new decryption routine that is capable of decrypting the virus, yet bears little or no resemblance to any prior decryption routine. Next, the virus encrypts this new copy of the virus body and mutation engine. Finally, the virus appends this new decryption routine, along with the newly encrypted virus and mutation engine, onto a new program.

As a result, not only is the virus body encrypted, but the virus decryption routine varies from infection to infection. This confounds a virus scanner searching for the tell-tale sequence of bytes that identifies
a specific decryption routine.

With no fixed signature to scan for, and no fixed decryption routine, no two infections look alike.The result is a formidable adversary.

The Scale of the Problem

The Tequila and Maltese Amoeba viruses caused the first widespread polymorphic infections in 1991.

In 1992, in a harrowing development, Dark Avenger, author of Maltese Amoeba, distributed the Mutation Engine, also known as MtE, to other virus authors with instructions on how to use it to build still more polymorphics.

It is now common practice for virus authors to distribute their mutation engines, making them widely available for other virus authors to use as if they were do-it-yourself kits.

Today, anti-virus researchers report that polymorphic viruses comprise about five percent of the more than 8,000 known viruses.

Of these, SARC reports only a small number of polymorphics “in the wild” — just 20 as of mid-1996. Yet this represents an increase of 25 percent from 16 polymorphics in the wild in mid-1995, a year earlier. Also, anti-virus researchers have identified 50 mutation engines. SARC reports 13 mutation engines in the wild as of mid-1996, up 30 percent in one year from 10 mutation engines reported in the wild as of mid-1995.

Two polymorphics — One Half and Natas — rank among the 20 most-prevalent computer viruses, according to the 1996 Computer Virus Prevalence Survey conducted by the National Computer Security Association (NCSA).

One Half slowly encrypts a hard disk. Natas, also known as SatanBug.Natas, is highly polymorphic, designed to evade and attack anti-virus software. It infects .COM and .EXE program files.

Polymorphic Detection

Anti-virus researchers first fought back by creating special detection routines designed to catch each polymorphic virus, one by one. By hand, line by line, they wrote special programs designed to detect various sequences of computer code known to be used by a given mutation engine to decrypt a virus body.

This approach proved inherently impractical, time-consuming, and costly. Each new polymorphic requires its own detection program. Also, a mutation engine produces seemingly random programs, any of which can properly perform decryption — and some mutation engines generate billions upon billions of variations.

Moreover, many polymorphics use the same mutation engine, thanks to the Dark Avenger and other virus authors who have distributed engines. Also, different engines used by different polymorphics often generate similar decryption routines, which makes any identification based solely on decryption routines wholly unreliable.

This approach also leads to mistakenly identifying one polymorphic as another. These shortcomings led anti-virus researchers to develop generic decryption techniques that trick a polymorphic virus into decrypting and revealing itself.

Generic Decryption

Generic decryption assumes:

- The body of a polymorphic virus is encrypted to avoid detection.
- A polymorphic virus must decrypt before it can execute normally.
- Once an infected program begins to execute, a polymorphic virus must immediately usurp controlof the computer to decrypt the virus body, then yield control of the computer to the decrypted virus.

A scanner that uses generic decryption relies on this behavior to detect polymorphics. Each time it scans a new program file, it loads this file into a self-contained virtual computer created from RAM. Inside this virtual computer, program files execute as if running on a real computer.

The scanner monitors and controls the program file as it executes inside the virtual computer. A polymorphic virus running inside the virtual computer can do no damage because it is isolated from the real computer.

When a scanner loads a file infected by a polymorphic virus into this virtual computer, the virus decryption routine executes and decrypts the encrypted virus body. This exposes the virus body to the scanner, which can then search for signatures in the virus body that precisely identify the virus strain. If the scanner loads a file that is not infected, there is no virus to expose and monitor. In response to nonvirus behavior, the scanner quickly stops running the file inside the virtual computer, removes the file from the virtual computer, and proceeds to scan the next file.

The process is like injecting a mouse with a serum that may or may not contain a virus, and then observing the mouse for adverse affects. If the mouse becomes ill, researchers observe the visible symptoms, match them to known symptoms, and identify the virus. If the mouse remains healthy, researchers select another vial of serum and repeat the process.

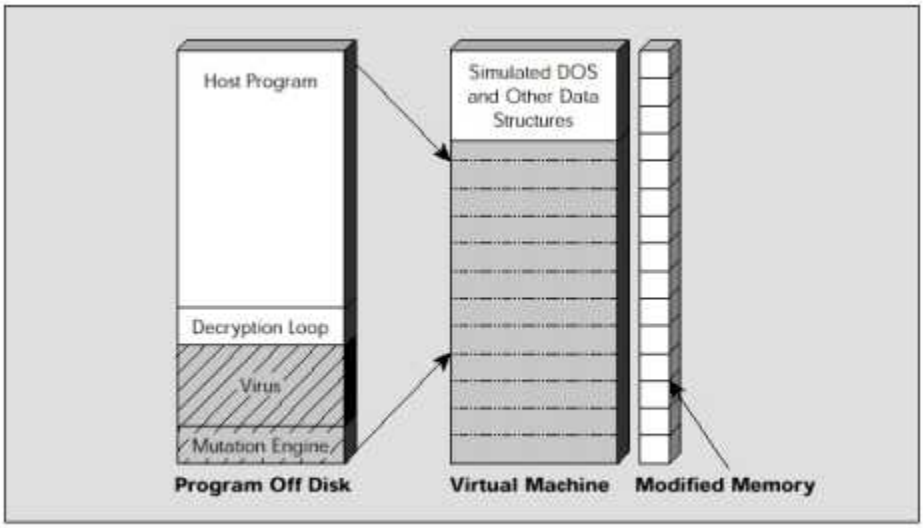

Figure 5. The generic decryption engine is about to scan a new infected program.

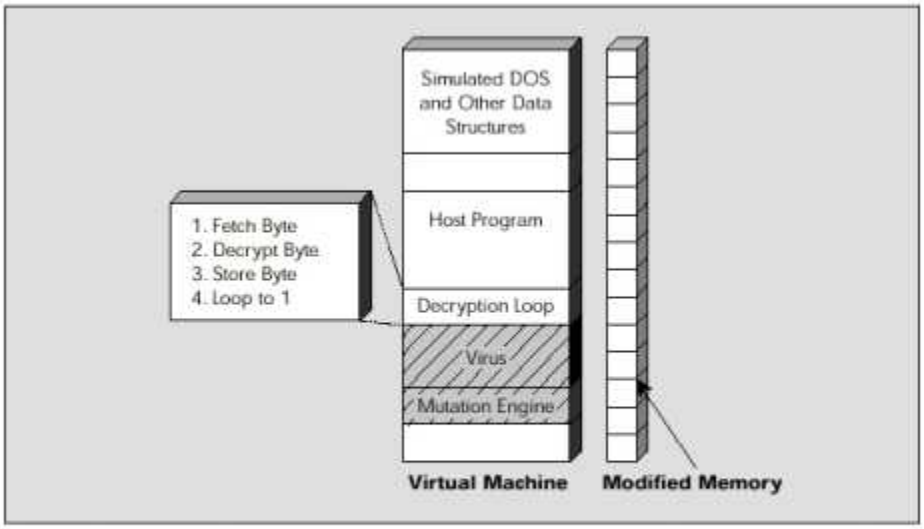

Figure 6. The generic decryptor loads the next program to scan into the virtual machine. Notice that each section of memory in the virtual machine has a corresponding modified memory cell depicted on the right-hand side of the virtual machine. The generic decryption engine uses this to represent areas of memory that are modified during the decryption process.

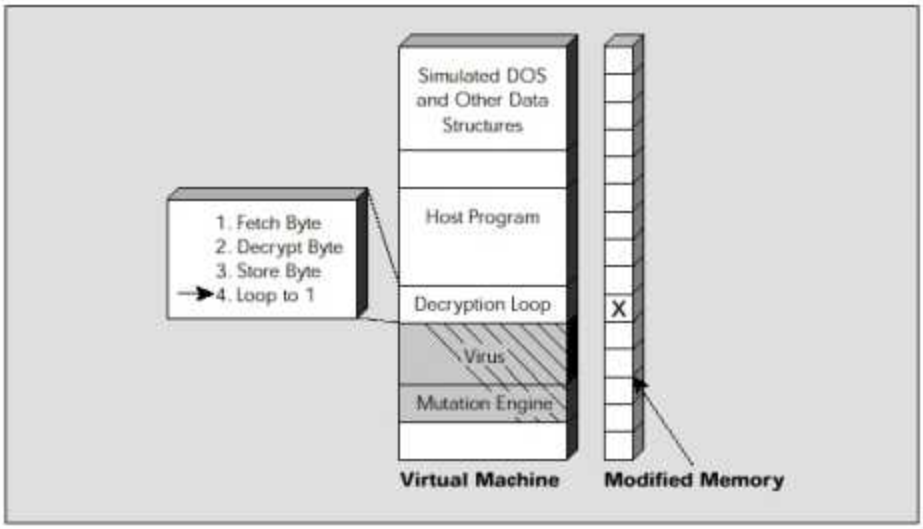

Figure 7. At this point the generic decryption engine passes control of the virtual machine to the virus and the virus begins to execute a simple decryption routine. As the virus decrypts itself, the modified memory table is updated to reflect the changes to virtual memory.

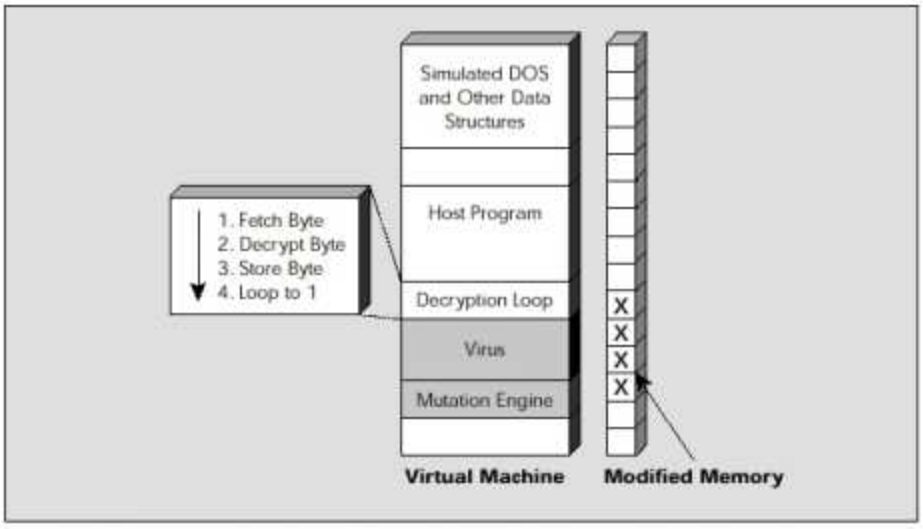

Figure 8. Once the virus has decrypted enough of itself, the generic decryption engine advances to the next stage.

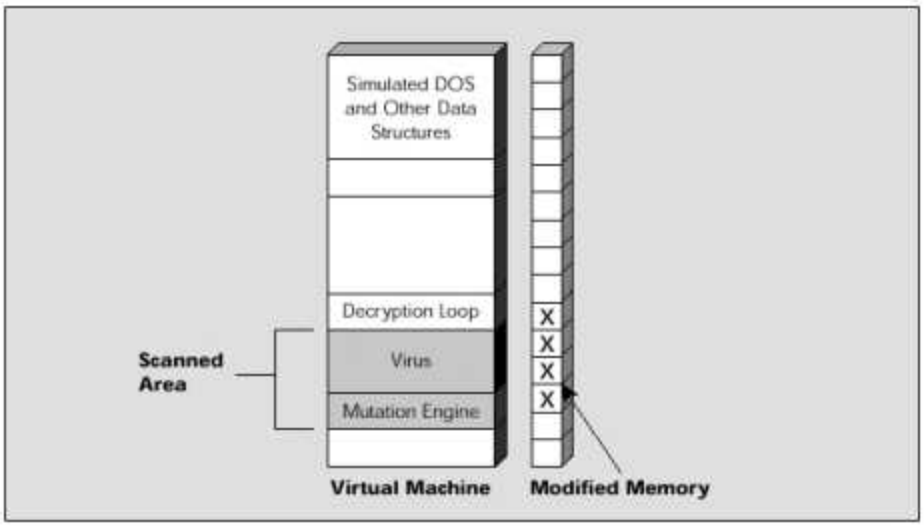

Figure 9. Now the generic decryption scanner searches for virus signatures in those areas of virtual memory that were decrypted and/or modified in any way by the virus. This is the most likely location for virus signatures.

The key problem with generic decryption is speed. Generic decryption is of no practical use if it spends five hours waiting for a polymorphic virus to decrypt inside the virtual computer. Similarly, if generic decryption simply stops short, it may miss a polymorphic before it is able to reveal enough of itself for the scanner to detect a signature.

Heuristic-Based Generic Decryption

To solve this problem, generic decryption employs "heuristics," a generic set of rules that helps differentiate non-virus from virus behavior.

As an example, a typical nonvirus program will in all likelihood use the results from math computations it makes as it runs inside the virtual computer. On the other hand, a polymorphic virus may perform similar computations, yet throw away the results because those results are irrelevant to the virus. In fact, a polymorphic may perform such computations solely to look like a clean program in an attempt to elude the virus scanner.

Heuristic-based generic decryption looks for such inconsistent behavior. An inconsistency increases the likelihood of infection and prompts a scanner that relies on heuristic-based rules to extend the length of time a suspect file executes inside the virtual computer, giving a potentially infected file enough time to decrypt itself and expose a lurking virus.

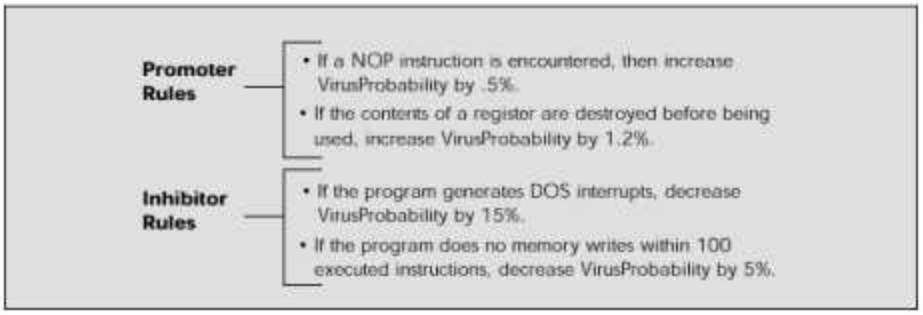

Figure 10. Initially the generic decryptor assumes that every file has a 10% probability of infection. Emulation continues as long as the virus probability is greater than zero. This virus probability is updated as the various rules observe virus-like or non-virus-like behavior during emulation.

Unfortunately, heuristics demand continual research and updating. Heuristic rules tuned to detect 500 viruses, for example, may miss 10 of those viruses when altered to detect 5 new viruses.

Also, as virus writers continue trying to make viruses look like clean programs, heuristics can easily balloon to the point where almost any program might share attributes that trigger the scanner to lengthen the time it takes to examine a file.

In addition, generic decryption must rely on a team of anti-virus researchers able to analyze millions of potential virus variations, extract a signature, then modify a set of heuristics while also guarding against the implications of changing any heuristic rules. This requires extensive, exhaustive regression testing. Without this commitment, heuristics quickly becomes obsolete, inaccurate, and inefficient.

The Striker System

Symantec's Striker system provides anti-virus researchers with a new weapon to detect polymorphics.

Like generic decryption, each time it scans a new program file, Striker loads this file into a selfcontained virtual computer created from RAM. The program executes in this virtual computer as if it were running on a real computer.

However, Striker does not rely on heuristic guesses to guide decryption. Instead, it relies on virus profiles or rules that are specific to each virus, not a generic set of rules that differentiate nonvirus from virus behavior.

When scanning a new file, Striker first attempts to exclude as many viruses as possible from consideration, just as a doctor rules out the possibility of chicken pox if an examination fails to detect scabs on a patient's body.

For example, different viruses infect different executable file formats. Some infect only .COM files. Others infect only .EXE files. Some viruses infect both. Very few infect .SYS files. As a result, as it scans an .EXE file, Striker ignores polymorphics that infect only .COM and .SYS files. If all viruses are eliminated from consideration, then the file is deemed clean. Striker closes it and advances to scan the next file.

If this preliminary scan does not rule out infection, Striker continues to run the file inside the virtual computer as long as the behavior of the suspect file is consistent with at least one known polymorphic or mutation engine.

For example, one polymorphic virus is known to perform math computations and throw away the results. A second polymorphic may never perform such calculations. Instead, it may use specific random instructions in its decryption routine. A third polymorphic may call on the operating system as it decrypts.

Striker catalogs these and nearly 500 other characteristics into each virus profile, one for each polymorphic and mutation engine.

Consider a set of generic heuristic rules that identify A, B, C, D, and E as potential virus behaviors. In contrast, a Striker profile calls for Virus 1 to execute behaviors A, B, and C. As it decrypts, Virus 2 executes behaviors A, B, and D, while Virus 3 executes behaviors B, D, and E.

If Striker observes behavior A while running a suspect file inside the virtual computer, this is consistent with viruses 1 and 2. However, it is not consistent with Virus 3. Striker eliminates Virus 3 from consideration.

The heuristic-based system must continue searching for all three viruses, however, because it observes behavior that is consistent with its generic rules.

If Striker next observes behavior B, this is consistent with viruses 1 and 2. Striker must continue scanning for these two viruses. However, the heuristics again continue to search for all three viruses.

Finally, if Striker observes behavior E, this eliminates Virus 2 from consideration, and Striker now pursues a single potential virus.

The heuristic-based scanner continues to search for all three viruses.

Under Striker, this process continues until the behavior of the program running inside the virtual computer is inconsistent with the behavior of any known polymorphic or mutation engine. At this point, Striker excludes all viruses from consideration.

On the other hand, a heuristic-based system scans for all viruses all the time. It must find some behavior inconsistent with all behaviors.

Striker's Strategic Advantages

Clearly the first advantage to Striker's approach is speed. The profiles enable Striker to quickly exclude some polymorphic viruses and home in on others. In contrast, heuristics labor on, scanning all program files against all available generic rules of how all known polymorphics and all known mutation engines might behave.

The profiles also enable Striker to process uninfected files quickly, minimizing impact on system performance. In contrast, heuristic-based scanning is more likely to decrease system performance, because uninfected files must also be scanned against all generic rules for how all known polymorphics and mutation engines might behave.

Second, anti-virus researchers are no longer forced to rewrite complex heuristic rules to scan for each new virus, then exhaustively test and retest to ensure they do not inadvertently miss a polymorphic the software previously detected.

Third, with Striker, a team of anti-virus researchers may work in parallel, building profiles for many new polymorphic viruses, swiftly adding each to Striker. Each profile is unique, much like a virus signature, independent of any other profile. The old profiles still work, and the new profile does not affect the old. Exhaustive, time-consuming regression testing is no longer necessary. It becomes easy to update anti-virus software by compiling new virus profiles into the Norton Antivirus database file that is posted online monthly or obtained on floppy disk.

Outlook

To date, generic decryption has proved to be the single most effective method of detecting polymorphics. Striker improves on this approach.

Yet it is only a matter of time before virus authors design some new, insidious type of virus that evades current methods of detection.

Virus authors might design a polymorphic virus that decrypts half the time, for example, yet remains dormant at other times. Anti-virus software could not reliably detect such a virus if it does not decrypt itself every time the file is loaded into the virtual computer. In this case, a hand-coded detection routine will be needed.

Or, imagine a host program that waits for the user to press a specific key and then terminates. A polymorphic infecting this host might only take control just after the user enters the required key-stroke. If the user enters the keystroke, the virus runs. If not, the virus gets no opportunity to launch.

However, inside the virtual computer created by generic decryption, the program would never receive the needed keystroke — and the virus would never have a chance to decrypt.

A small number of viruses are already resistant to detection by generic decryption. There's no doubt that virus authors will continue to design new viruses, using new technologies, creating new problems. Anti-virus researchers will need to deal with these new threats, just as Striker today delivers the solution that best protects computer users against polymorphics.
