Understanding encryption and polymorphism

Escalation is a good word to use here.
Virus programmers may encrypt messages so they can not be easily seen. In the same way many viruses contain encrypted code to hide what they do. Before there were virus scanners, there were programs written to detect possible Trojans. One such program was written by Andy Hopkins in 1984 and was called CHK4BOMB. When you used it to check out a program, it would alert you to anything suspicious in the program, like direct disk writes and formatting, as well as print out any messages it found. Obviously, a fully encrypted program, even one that did and said nasty things, would look safe on examination.

Yet, encrypted viruses are not complete encrypted. Encrypted code is no longer executable code--it simply won't run. For an encrypted virus to actually run, it has to decrypt its code and data. The portion that does this decryption is not encrypted because it has to run. This portion is refered to as a decryptor.

Encryption techniques

Some viruses use very simple encryption techniques such as incrementing, decrementing, or rotating each byte in the code. They may also negate or logically not each byte. Such encryption does not require an encryption key--a additional value used in encrypting each byte or word (two bytes). Techniques that use a key include adding, subtracting and xoring. A key value can also be used in rotating a byte. Additionally, keys themselves come in three types.
A static key is one that doesn't change as the virus uses it--it is a set value. Viruses using a static key might add 128 to each byte, rotate each byte 3 places to the right, or xor each word with 0F8F8h.

A variable key is where the key value varies in some way. This key starts as a static value and is then modified during the decryption. The key may itself be incremented, decremented, xored, rotated, etc.

Both static and changing keys produce predictable results. Specifically, the resulting encrypted code looks the same in every replication of the virus. Therefore, if you used a simple string scanner with a string from within the encrypted portion of the virus, you would still detect all its parents and progeny. Such encryption presents no problem to the antivirus industry. But the third type of key does.

A random key is one that changes from infection to infection. Cascade, for example, bases its key on the size of the host file--which obviously changes a lot. Other viruses use a pseudo-random key, such as fetching, storing, and using the current timer tick count, or the current 100ths of a second value. Any of these approaches produces a virtually random and unpredictable key.

This causes problems for those write programs that detect viruses. Since the code and data in such a virus changes radically, string scanning product developers must chose a string from the the only part of the virus that doesn't change--the decryptor. Early on this lead to two major problems in the industry.

The first problem involved false alarms. Early grunt scanners (scanners that examine an entire file) that used the same string for Cascade would detect each other as being infected. This problem was solved by encrypting strings.

The second problem involved copyright. Some early product developers claimed copyright on their scan strings, which, when you think about it, means they were copyrighting fragments of another programmer's code--the virus programmer's code. Ross Greenburg, the developer of Flu-Shot and VirexPC, had a request out for virus strings. As Ross tells it, someone downloaded a bunch of strings, sent them to him, and he used them. Unfortunatly, those strings had been extracted from McAfee's scanner. McAfee threatened a lawsuit, but never carried out the threat.

Herein lies the problem. What then about a randomly encrypted virus with a short decryptor? In the Fish virus, for example, there are only 14 usable bytes. So string scanning products virtually have to use the same pattern, do they not? How then can one company claim a copyright on a string many others are forced to use also?

Virus Bulletin regularly publishes search strings and the Fish virus byte pattern can be found in the July, 1991 issue. Here it is reprinted:

```
E800 005B 81EB A90D B958 0D2E 8037
```

By the way, I did not ask permission to reprint this. So is my printing this pattern a violation of VB copyright?

Virus Bulletin itself answers "No" and points how ludicrous this idea is:

"Some misunderstandings have arisen in the past about the copyright notice which appears at the foot of each page of the bulletin; does this notification apply equally to hexadecimal search patterns? The answer, of course, is an empahtic NO - search patterns are not intellectual property or original material and are beyond copyright. There have been incidents in the United States of software developers threatening lawsuits against other software developers on the basis that search patterns have been 'stolen'.

"The VB Table of Known IBM PC Viruses is designed to be actively used; the patterns are supplied to help systems engineers with diagnosis but may also be used in the development of comprehensive scanning software. Use of these patterns is positively to be encouraged."

But encryption, even random key encryption and short decryptors are truely not a problem to antivirus developers when it comes to detection. The real problem is polymorphism.

Polymorphism

Since a string scanner can only detect randomly encrypted viruses by using their decryptor, what happens if the decryptor itself changes with each infection?

"Scanning can't find all viruses." Was reportedly the premise of two virus researchers in the United States.

According to sources such as *Virus Bulletin*, in January of 1990 each of these men sent out a virus to prove their claim. Patrick Toulome sent his Virus-101 to the developer of a scan product. Mark Washburn sent out his V2P1 or Chameleon virus. These were the first two polymorphic viruses.

When Toulome's virus went beyond the researcher he sent it to, he didn't appreciate it. He stopped making viruses. Washburn, on the other hand, made and released several more--each progressively more polymorphic.

The general meaning of polymorphic is "having many forms" and could thus be applied to any randomly encrypted virus--since they indeed have many forms. However, the use of this word in antivirus research and product development, as well as our use here, is more specific.

A polymorphic virus is a randomly encrypted virus that is also programmed to randomly vary its decryption routine. Thus the decryptor itself has "many forms"--is polymorphic.

Before February of 1991 there were several terms used to describe these viruses: mutating, garbling, self-modifying, variably decrypting, and such. In that month, however, Fridrik Skulason and Alan Solomon coined "polymorphic" as it is applied to these viruses. The term caught on quickly.

Now that we've explained the the definition and history of the term, polymorphic, we're going to look at what it really means. But be warned. This portion of our discussion of viruses gets more analytical in nature and thus, necessarily, more technical.

During 1990 four polymorphic viruses were developed by Dark Avenger, based on his V800 virus. In an interview with Sarah Gordon, Dark Avenger said "Proud, Evil, Phoenix,are variants of one virus." This may mean that the fourth, Phoenix.1226, was the first programmed. None of these are in the wild, but we'll use the 1226 version here as an example of polymorphism.

The decryption routine for phoenix.1226 is 32 bytes long. Within that 32 byte routine, 18 bytes are variable. This variation is accomplished in two ways.

First off, two of the bytes can each have one of two values, these bytes represent to two conditional jumps that can either be a jns (jump if not sign) instruction, with a byte value of 79h, or jge (jump if greater than or equal) instruction, with a byte value of 7Dh.

The remainder of the variability is more complex. There are five processor registers used in the decryptor. The first two used have to be pointer registers since they are used in indirect memory addressing. This limits the available registers to bx, di, and si (bp is not used). The other three registers are used for storage and may be selected from ax, bx, cx, or dx. Also, if bx was used as a pointer than either di or si, whichever is available, can be used.

Here are the first 18 bytes of the decryption routine. The numbers on the left are the byte values (with ?? being a variable byte), while p1 and p2 are the pointer registers, and s1 and s2,are the general storage registers.

```
xchg  bp, ax
mov   p1, 0100
inc   p1
add   p1, [p1]
mov   p2, p1
xor   s1, s1
mov   s2, 0254
push  s2
xor   s1, [p1+22]
```

Represented as a search string this becomes:

```
95?? 0001 ??03 ??8B ??33 ???? 5402 ??33 ??22
```

Here, nearly half the bytes are variable. Moreover, the values 54 and 02 differ in other phoenix variants. While no simple search string is possible for this virus, string scanners can detect it easily and reliably. This involves the concept of wildcard searching. Something you're probably familiar with.

You use a wildcard when you type "dir *.doc" at the DOS prompt. This returns all the files with the "doc" extention. You could also type "dir *.d??" or "dir *.d*" to get the "doc" as well as the "dbf" files. You may also use a query language. If so, then you know that searching for "wom?n" will find all occurances of "woman" and "women". In a similar fashion, a string scanner can recognize wildcards. Thus, given our string above, the scan engine would simply match the bytes given a value and ignore whatever byte was in the virus where its detection string has a wildcard.

Remember escalation?

Alan Solomon is well known to talk to himself in his presentations. No, he's not schizophrenic. He acts out the roles of a virus programmer and an antivirus programmer engaged in a development arms race--each trying to outdo the other. (During such presentations Alan literally changes between a white hat and a black hat so the audience knows which Alan is speaking.)

Virus programmer: "I'll make a decryptor and pad the code with variable numbers of do-nothing instructions."

Antivirus programmer: "Fine. I'll make variable wildcards that can slide ahead looking for the next real byte."

Virus programmer: "Oh yeah. Well I'll also vary the order of the real bytes in my decryptor."

Antivirus programmer: "Well then I guess I'll just slide clear through and look for each real byte individually."

Virus programmer: "Oh yeah. So I'll use different instructions to produce the same results."

Antivirus programmer: "Then I'll have to look for all those different instructions when I scan the decryptor."

Virus programmer: "Then I'll plug in an advanced table driven engine that creates billions of different decryptor sequences."

Antivirus programmer: "I'll change over to an emulation driven engine that analyzes the decryption results."

Get the idea?

False positives

Of course neither virus nor antivirus technology is nearly that simple. But the escalation has, well, escalated--on both sides. Yet, detection is not a dilema for antivirus developers. This remains fairly straight forward given the "fuzzy logic" type of rule-based systems and emulators we have. The real problem--which, by the way involves you directly--is not in detecting 100 percent of the samples of a polymorphic virus. The problem is, rather, detecting 101 percent of the samples.
How do you detect 101 percent? It's easy--too easy actually. You just detect 100 percent of the samples of a given virus, plus you detect that virus in 1 percent of other programs--clean, uninfected programs. Detecting a virus where there is none is called a false positive, or false alarm. The less detectable code there is in a virus, plus the fuzzier your sliding and sequencing logic becomes, the greater your changes are of thinking something else is that virus. Well, it certainly looks like that virus.

Take MtE for example.

The MtE is the Mutation Engine programmed and distributed by Dark Avenger. It is not a virus. It comes as a linkable object file to add to your own virus. The engine generates a polymorphic decryptor for your virus. It also comes with instructions and a sample virus. I first saw the virus back when I was with Certus, working on Novi.

Evidently, it took Dark Avenger several months to program the engine. It took me most of an afternoon to write a detector. Most developers I've spoken with also had a detector in very short order. That detector was for Novi and, while it would detect the MtE reliably right from the start, it took several reworkings to get rid of the false positives. Interestingly, just after Novi shipped with that detection, Borland shipped a C++ with a program that Novi thought was infected. Later when I was with Symantec, and we were developing Norton AntiVirus 3.0, both my Novi MtE detector and the one in NAV 2.1 were scrapped for something more reliable; which later also had some false ids (Sigh).

Since MtE, other polymorphic engines have also come out. These include CLME, DAME, DSCE, GPE, MGEN, NED, RDA, SMEG, TPE, VICE, and several others.

With products looking for more and more polymorphic viruses, there is an impact on product speed. Although many products have added emulation engines to cope with polymorphics, even these can slow things down a lot.

So the main impact of encryption and polymorphism on you, the user, is twofold. Slower scanners and false alarms.

Therefore, be prepared. As the virus and antivirus camps continue escalation, you may need to deal with these issues--on your personal or company computers. We hope that an understanding of virus encryption and polymorphism will help you cope.