The Evolution of Polymorphic Viruses

THE EVOLUTION OF POLYMORPHIC VIRUSES
Fridrik Skulason

The most interesting recent development in the area of polymorphic viruses is how limited their development actually is. This does not mean that there are no new polymorphic viruses, far from it-new ones are appearing constantly, but there is nothing 'new' about them-they are just variations on old and well-known themes.

However, looking at the evolution of polymorphic viruses alone only shows half of the picture-it is necessary to consider the development of polymorphic virus detection as well. More complex polymorphic viruses have driven the development of more advanced detection methods, which in turn have resulted in the development of new polymorphic techniques.

Before looking at those developments that can be seen, it is perhaps proper to consider some basic issues regarding polymorphic viruses, starting with the question of why they are written.

That question is easy to answer-they are written primarily for the purpose of defeating one particular class of anti-virus product-the scanners. Considering virus scanners are the most popular type of anti-virus program, it is not surprising that they are the subject of attacks.

At this point it is worth nothing that polymorphic viruses pose no special problems to a different class of anti-virus product, namely integrity checkers. This does not mean that integrity checkers should be considered superior to scanners-after all there is another class of viruses, the 'slow' viruses, which are easily detected by scanners, but which are a real problem for integrity checkers.

Fortunately, polymorphic slow viruses are not common at the moment. As a side note 'slow polymorphic' viruses also exist, and should not be confused with 'polymorphic slow' viruses. This category will be described at the end of this paper, together with some other 'nasty' tricks.

Considering how virus scanners work, a virus author can in principle attack them in two different ways-either by infecting an object the scanner does not scan, or by making the detection of the virus so difficult that the scanner, or rather the producers of the scanner may not be able to cope with it.

Polymorphic viruses attempt to make detection difficult-either too time consuming to be feasible, or beyond the technical capabilities of the anti-virus authors.

The success of virus authors depends not only on their programming skills, but also on the detection techniques used. Before describing the current techniques, however, a brief classification of polymorphic viruses is in order.

1) Encrypted, with variable decryptors. This is the largest and currently the most important group. Several methods to implement the variability are discussed below, but most of them should be familiar to readers of this paper.

2) 'Block-swapping' Viruses. Only a handful of viruses currently belong to this group, but they demonstrate that a polymorphic virus does not have to be encrypted. These viruses are composed of multiple blocks of code, theoretically as small as two instructions, that can be swapped around in any order, making the use of normal search strings nearly impossible.

3) Self-modifying viruses using instruction replacement techniques. This is where the virus may modify itself by replacing one or more instructions in itself with one or more functionally equivalent instruction when it replicates. So far this category is only a theoretical possibility, as no viruses have yet been written that use this technique. It is possible that some such viruses will appear in the future, perhaps only to written to demonstrate that it can indeed be done.

Considering that the viruses that currently fall into the second group are easy to detect using ordinary search strings, and that the third group is non-existent, the only polymorphic viruses currently of interest are encrypted ones.

For that reason the term 'polymorphic viruses' should, in the rest of this paper, really be understood to mean only viruses of the first group, that is, encrypted with variable decryptors.

Basically the detection methods fall into two classes-those that detect and identify only the decryptor and those that look 'below' the decryptor, detecting the actual virus. This is not a strict 'either-or' classification- a scanner may analyse the decryption loop to determine that it might have been generated by a particular virus, before spending time decrypting the code.

There are several different methods that have been used to detect and identify decryption loops-which used to be the standard way of detecting polymorphic viruses-but there are several significant problems with these methods. The most common methods are described later, but if they are only used as the first step, and the virus then properly decrypted some of the following problems disappear:

- Virus-specific. Basically, the detection of one polymorphic virus does not make it any easier to detect another.

- More likely to cause false positives. As we get more and more polymorphic viruses, capable of producing an ever-increasing variety of decryptors, the chances of generating a false positive increase, as some innocent code may happen to look just like a possible decryptor.

- Identification is difficult. Many polymorphic viruses will generate similar decryptors, and it is entirely possible that a scanner will mis-identify a decryptor generated by one polymorphic virus as having been produced by another, unrelated virus. Also, in the case of variants of the same polymorphic virus, it may be possible to determine the family, but not the variant.

- No disinfection. Virus disinfection requires the retrieval of a few critical bytes from the original host file that are stored usually within the encrypted part of polymorphic viruses. This means that virus- specific disinfection is generally not possible, at it would require decrypting the virus.

On the positive side, detection of a particular decryptor may be quite easy to add, althought that depends on the design of the scanner and the complexity of the virus. The decryption techniques are old, and several anti-virus producers have abandoned them, in favour of more advanced methods.

- Search strings containing simple wildcards
- Search strings containing variable-length wildcards
- Multiple search strings
- Instruction usage recognition
- Statistical analysis
- Various algorithmic detection methods

The limitation of this method are obvious, as it can only handle a few 'not very polymorphic' viruses, which are sometimes called 'oligomorphic'. They may for example make use of a simple decryption loop, with a single variable instruction. The last variable polymorphic virus uses two different instruction, NEG and NOT, which differ by only one bit. Defeating this detection method is easy: just insert a random number of 'junk' instructions at variable places in the code. 'Junk' does not have to mean 'invalid', but rather any instruction that can be inserted in the decryption loop without having an effect. Typical examples include NOP, JMP $+2, MOV AX,AX and other similar 'do nothing' instructions.

This method takes care of decryptors that contain those junk instructions. However, there are two problems with this approach. Some scanners cannot use this method as their design does not allow variable-length wildcards, but that really does not matter, as the technique is very easy to defeat: just make the decryptor slightly more variable so that no single search string, even using a variable-length wildcard will match all instances of the decryptor. This can be done in several ways.

- Changing register usage: For example the DI register might be used for indexing, instead of SI, or the decryption key might be stored in BX instead of AX
- Changing the order of instructions: If the order of instructions does not matter, they can be freely swapped around.
- Changing the encryption methods: Instead of using XOR, the virus author could just as well use ADD or SUB.

This is generally considered an obsolete technique, but many anti-virus producers used it back an 1990 when the Whale virus appeared. This virus could be reliably detected with a fairly large set of simple search strings. Today, however, most of them would probably use a different method. This detection method can easily be defeated by increasing the variability of the decryptor past the point where the number of search string required becomes unreasonably large. There are other cases where the multiple search string technique has been used. One anti-virus company had access to the actual samples of a particular polymorphic virus that were to be used in a comparative product review. Rather than admitting that they were not able to detect the virus, they seem to have added a few search string to detect those particular samples-and they did indeed score 100% in that test, although later examination revealed that they only detected 5% of the virus in question.

This method was developed to deal with Dark Avenger's Mutation engine. It basically involved assuming initially that all files are infected, then tracing through the decryptor, one instruction at a time. If an instruction is found that could not have been generated by a particular virus as a part of the decryptor, then the virus is not infected by that virus. If one reaches the end of the decryptor, still assuming that the file is infected, it is reported as such. There are two major ways to attack this technique, but the more obvious is to increase the number of possible instructions used in the decryptor. If a virus used every possible instruction in a decryptor, it simply could not be detected with this method without modifying it. The second method is more subtle, but it involves making it more difficult to determine when the end of the decryption loop has been reached.

This method is generally not used, due to the unacceptably large risk of false positives. It basically involves statistical analysis of the number of certain in the decryptor. It works best with viruses that generate large decryptors, that uwe few and uncommon 'do-nothing' instructions.

Other algorithmic detection methods are possible, and are frequently used. Sometimes they are inly used to quickly eliminate the possibility of a particular file being infected with a particular virus, for example:

It should be obvious from this example that the rules can get complex, perhaps unreasonably complex, and obviously require significant work to implement. Also, in some instances it is just not possible to get a sufficient number of rules like this to ensure accurate detection, not even considering the rules the virus itself may use to determine if a file has already been infected as the number of false positives would be too high.

At this point it is very important to bear in mind that, while false positives are a very serious problem for the anti-virus author, they do not matter at all to the virus author. A false positive just means that the virus will not infect one particular file it might otherwise have infected... so what-after all, it has plenty of other files to infect.

Having looked at the detectors that only detect the decryption loop, we must look at the more advanced detectors, which detect the actual virus, instead of just the encryption loop.

Compared to the decryptor-detecting methods, the following differences are obvious:

- More generic. These methods require significantly more initial work, but the extra effort required to add detection of a new polymorphic virus is far less than with some of the other methods described above.

- Less chances of false positives. Having decrypted the virus, it should be possible to reduce the chances of false positives almost down to zero, as the entire virus body should be available.

- Identification is easy. When the virus has been decrypted, identification is no more difficult than in the case of non-encrypted viruses.

- Easy disinfection. The sample applies to disinfection-it should not be any more difficult than if the virus had not been encrypted to begin with.

There are two such techniques which have been used to detect polymorphic viruses.

The X-raying technique was probably only used in two products, both of which have mostly abandoned it by now. It basically involved assuming that a particular block of code had been encrypted with an unknown algorithm, and then deriving the encryption method and the encryption keys from a comparison between the original and encrypted code.

Assume that manual decryption of one virus sample reveals that a particular block of code should contain the following byte sequence:

B8 63 25 B9 88 01 CD 21

The corresponding encrypted block of code in a different sample looks like this:

18 C4 8B 0C 34 C2 07 F0

Is there any way this sequence could have been obtained from the first one by applying one or two primitive, reversible operations like for example:

XOR with a constant
ADD/SUB with a constant
ROL/ROR a fixed number of bytes

Yes, because XORing the two sequences together generates the sequence:

A0 A7 AE B5 BC C3 CA D1

Calculating the differences between the bytes in that sequence give the following result:

07 07 07 07 07 07 07 07

which shows that the original sequence, and (presumably) the entire virus body can be obtained by XORing each byte with a key, and then adding the constant value of 7 to that key, before applying it to the next byte.

Using this method, it may be possible to deduce the operation of the decryptor, without looking at it at all. There is a variant of the X-ray method which has been developed by Eugene Kaspersky, which works in a different way, but produces the same result.

The reason 'X-raying' has mostly been abandoned is that it can easily be defeated, for example by using an operation the X-ray procedure may not be able to handle, by using three or more operations on each decrypted byte or by using multiple layer of encryption.

The last method to be developed does not suffer from that limitation, and can handle decryptors of almost any complexity. It basically involves using the decryptor of the virus to decrypt the virus body, either by emulating it, or by single-stepping through it in a controlled way so the virus does not gain control of the execution.

- Which processor should be emulated? It is perfectly possible to write a virus that only works properly on one particular processor, such as a Cyrix 486 SLC, but the decryptor will just generate garbage if executed on any other processor. An intelligent emulator may be able to deal with this, but not the 'single-stepping' method.

- Single-stepping is dangerous - what if the virus author is able to exploit some obscure loophole, which allows the virus to gain control. In this case, just scanning an infected file would result in the virus activating, spreading and possible causing damage, which is totally unacceptable. It should be noted that a very similar situation has actually happened once-however the details will not be discussed here.

- Emulation is slow-if the user has to wait a long time while the scanner emulates harmless programs, the scanner will probably be discarded, and obviously a scanner that is not used will not find any viruses.

- If the virus decryptor goes into an infinite loop and hangs when run, the generic decryptor might do so too. This should not happen, but one product has (or used to have) this problem.

- How does the generic decryptor determine when to stop decrypting code, and not waste unacceptable amount of time attempting to decrypt normal, innocent programs?

- What if the decryptor includes code intended to determine if it is being emulated or run normally, such as a polymorphic timing loop, and only encrypts itself if it is able to determine that it is running normally?

- What if the decryptor is damaged, so that the virus does not execute normally? A scanner that only attempted to detect the decryptor might be able to do so, but a more advanced scanner that attempts to exploit the decryptor will not find anything. This is for example the case with one of the SMEG viruses-it will occasionally generate corrupted samples. They will not spread further, but should a scanner be expected to find them or not?

Finally, it should be noted that there are other ways to make polymorphic viruses difficult than just attacking the various detection techniques as described above.

'Slow polymorphic' viruses are one such method. They are polymorphic, but all samples generated on the same machine will seem to have the same decryptor. This may mislead an anti-virus producer into attempting to detect the virus with a single search string, as if it was just a simple encrypted but not polymorphic virus.

However, virus samples generated on a different machine, or on a different day of the week, or even under a different phase of the moon will have different encryptors, revealing that the virus is indeed polymorphic.

Another recent phenomena has been the development of more 'normal-looking' polymorphic code. Placing a large number of 'do-nothing' instructions in the decryptor may be the easiest way to make the code look random, but it also makes it look really suspicious to an 'intelligent' scanner, and worthy of detailed study. If the code looks 'normal', for example by using harmless-looking 'get dos-version number' function calls, it becomes more difficult to find.

So, where does this leave us? Currently anti-virus producers are able to keep up with the virus developers, but unfortunately the best methods available have certain problems-the one most obvious to users is that scanners are becoming slower. There is no indication that this will get any better, but on the other hand there are no signs that virus authors will be able to come up with new polymorphic techniques which require the development of a new generation of detectors.