Polymorphic Generators

Polymorphic Generators

Polymorphic viruses

The rise of polymorphic viruses can be seen as virus writers' response to the increasing expertise of virus scanners. Since properly built scanners can recognise viruses by their characteristic code, the obvious way to try to beat scanners was to design viruses that change their code, thus rendering recognition with search strings impossible.

Polymorphic viruses employ code alteration and encryption to hide themselves from scanners. Their usual tactic is to encrypt the main part of their code with a variable key and leave only the decryption executor unencrypted. The decryption code is altered during every infection to prevent detection with a search string.

However, it takes considerable skill to design a polymorphic virus. This kept the number of true polymorphic viruses quite small for a relatively long time. Of course, this couldn't last forever: At some stage, the heavyweights of the virus trade took notice and came to rescue their less skilled brethren by writing and distributing polymorphic generators.

Polymorphic generators

Polymorphic generators are routines which can be linked to existing viruses. The generators are not viruses per se; their purpose is to hide actual viruses under the cloak of polymorphism.

The first all-purpose polymorphic generator was the Mutation Engine, or MtE. Published in 1991, capable of billions of different permutations, linkable to any virus, it heralded the age of instant polymorphism. Today, there are 33 different viruses which are known to use the MtE.

Other polymorphic generators followed in MtE's wake. The next two appeared late in the year 1992. They were the TridenT Polymorphic Engine (TPE) and NuKE Encryption Device (NED).

TPE was written in the Netherlands. In principle it is capable of producing smaller number of different permutations than the MtE. However, it created detection problems for antivirus products because the decryptors it creates are more generic than those produced by MtE. NuKE's generator wasn't quite as advanced, but unlike most other polymorphic generators, it was distributed as readable source code instead of an object module.

Other known polymorphic generators are Dark Angel's Multiple Encryptor (DAME), Darwinian Genetic Mutation Engine (DGME), Dark Slayer Mutation Engine (DSME), MutaGen, Guns'n'Roses Polymorphic Engine (GPE) and Dark Slayer Confusion Engine (DSCE).

These generators are typically distributed via underground networks, virus exchange BBSs and private areas in the internet.

Operating Principles

Polymorphic generators are code modules which a programmer can incorporate into a program. After this, the program can use the functions the code module contains. This process is called linking. Once a generator is linked to a virus, it becomes an intrinsic part of the said virus. The virus will thereafter carry the engine along while spreading itself.

It should be noted that the generator itself does not care in which kind of a program it is linked to. The known polymorphic generators are clearly written to be linked to viruses, but in principle they could be used in other kinds of programs as well.

When a virus that employs a polymorphic generator is infecting a program file (or some other object), it requests the generator to create an encrypted copy of the virus code and the generator itself. Besides performing the encryption, the generators also create a decryptor - a routine which is able to undo the encryption applied to the actual virus code.

The generators often use relatively simple encryption techniques. However, they do change the encryption key during every execution. This alone makes the detection of such a virus difficult, but encrypted viruses retain one Achilles heel: the decryption routine, which must remain unencrypted if it is to be executable. Thus, the true effectiveness of a polymorphic generator is measured by its ability to mutate the decryption routine.

All polymorphic generators need some kind of a randomisation routine in order to create different algorithms each time. Some of the generators allow the virus programmer to substitute his own randomisation routines instead of the original one.

Polymorphic generators are able to create completely different encryption methods and a wide variety of different decryption routines for them. They modify their decryption routines by such means as shifting the commands inside the routine around, adding ineffectual commands in random places and using different processor registers and opcodes.

The basic idea is to make the binary image of the decryption routine totally different between different infections. All this makes it impossible to search for the decryption routine with fixed search strings - there is no search string that could always be found in infections made by a polymorphic virus.

How does a virus using a polymorphic generator infect a file?

1. A clean file before the infection. We'll call this the victim file.

2. The virus starts the infection process by modifying the victim file's first commands. It replaces them with a command to jump to the end of the file. The original first bytes of the file are stored in the virus's body.

3. Next, the virus calls the polymorphic generator to create an encrypted copy of the virus code and the generator itself. The generator also creates a decryption routine, which is added to the end of the victim file.

4. The encrypted code is added to the end of the victim file. This encrypted section contains three parts: a copy of the actual virus code, the original first bytes of the victim file, and the code of the polymorphic generator.

Limitations

When the first polymorphic generators were found, it was feared that there would be a huge rise in the number of polymorphic viruses. However, these generators have not proved as popular as was originally thought - only about one hundred viruses are known to use a generator.

One of the reasons for this is that a generator must be linked to the program to be encrypted, and since the operation requires changes to the program itself, some programming experience is necessary. This alone places the generators out of the reach of the run-of-the-mill virus enthusiasts. Unfortunately, the generators usually come with detailed instructions on their use, so that virus aficionados with even limited experience of assembly programming can easily use them.

Another limitation is the generators' size. Although the generators are quite small in themselves, they do increase the size of viruses by some amount. This makes it difficult to link them to boot sector viruses, which have limited code space. No generator-masked boot sector viruses have been found. With the exception of V-Sign (a mildly polymorphic boot sector virus), polymorphic capabilities seem to be the privilege of file viruses.

Of course, the advantage that viruses get from polymorphic generators is somewhat questionable. If an anti-virus program is able to recognise the presence of a particular generator, it is usually able to detect all viruses masked by it.

Detection

Despite the cunning nature of polymorphic generators, viruses masked by them can be detected by using proper tools. Antivirus programs often employ algorithmic means to recognise files infected by polymorphically hidden viruses. Another way to find such viruses is to use checksumming. It is also possible to try to solve the encryption and search for the virus underneath the encryption layer.

Algorithmic methods

Algorithmic methods are based on the fact that however much a generator mutates the decryption routine, it must still contain certain programming structures which make the decryption possible. If a program file contains such structures, the antivirus program can say with sufficient certainty that the file is infected by a polymorphically cloaked virus.

As polymorphic generators vary a lot, a different algorithm is needed for each generator - and in order to build such algorithm, the generator will have to be studied closely.

However, the algorithmic methods have a certain weakness: they are prone to false positives. The program structures employed by polymorphic generators can be very random. This means that similar structures sometimes occur inside legitimate program code. False alarms may crop up especially if data files are also included in the search, because they typically contain data similar to the random 'garbage-code' which the generators produce. It is relatively easy to create an algorithm that will find all infections created with a polymorphic engine, but if the algorithm would also flag a large amount of clean programs as infected, it is useless.

Checksumming

Checksums are comparison values calculated from the executables in a system. These values are stored in a database. When a checksum search is made, the checksums are re-calculated and compared with the original values in the database. Since this method detects all changes to a system, the mutability of polymorphically hidden viruses does them no good; a change is a change, and thus detectable.

Checksumming has its drawbacks, too: checksummers suspect all changes that happen inside a system, and occasionally give warnings of ordinary programs which alter their own code. Nowadays, checksummers are usually equipped with an exclude-list and a heuristic faculty to prevent this from happening.

Although theoretically able to detect all changes to a system, checksummers are vulnerable to stealth viruses. If such a virus is active in a computer's memory, it is able to hide all the changes it has made. When stealth viruses are involved, checksummers base their calculations on false data, and will consequently find everything to be in order. It should be noted that polymorphic viruses which also stealth their presence are very rare, simply because they are technically difficult to create.

Decryption-based detection

The decryption-based detection of polymorphic viruses work by first reasoning whether the examined object is encrypted. If the object seems to warrant suspicion, generic decryption methods are applied to it, and a string-based search is done to the code found underneath the encryption.

This method works against some polymorphic generators with great success, but is difficult to implement for others.

What is the best solution?

Checksumming is the strongest method against polymorphic viruses - as long as the machine is clean when the checksummer is installed, and the virus is not falsifying the information received by the checksummer. Checksummers will also detect those polymorphic (and normal) viruses that have not yet been analysed.

The algorithm-based detection mechanisms against polymorphic viruses tend to have problems with false alarms, but these can be overcome by designing the detection engine carefully. One advantage of algorithm- based detection is that, once a detection engine is able to detect a certain polymorphic generator, it will probably detect all viruses utilising it.

A decryption-based detection mechanism can only detect those polymorphic viruses that have been analysed by the creator of the antivirus product, but it is very unlikely to produce false alarms. Furthermore, such a mechanism is also able to detect the exact variant of the virus in question - this is something that most algorithm-based detection methods are unable to do.
