VIRUS ANALYSIS

Striking Similarities

Frédéric Perriot, Peter Ferrie and Péter Ször
Symantec Security Response, USA

W32/Simile is the latest 'product' of the developments in metamorphic virus code. The virus was released in the most recent 29A #6 issue in early March 2002.

The virus was written by the virus writer who calls himself 'The Mental Driller'. Some of his previous viruses, such as W95/Drill (which used the Tuareg polymorphic engine), have proved very challenging to detect.

W32/Simile moves yet another step up the scale of complexity. The source code of the virus is approximately 14,000 lines of assembly code. About 90% of the virus code is taken up by the metamorphic engine itself, which is extremely powerful.

The virus was named 'MetaPHOR' by its author, which stands for 'Metamorphic Permutating High-Obfuscating Reassembler'.

The first generation virus code is about 32 KB and there are three known variants of the virus in circulation. Samples of the original variant which was released in the 29A issue have been received by certain anti-virus companies from some major corporations in Spain, indicating a minor outbreak.

W32/Simile is highly obfuscated and challenging to understand. The virus attacks disassembling, debugging and emulation techniques, as well as standard evaluation-based techniques for virus analysis. In common with many other complex viruses, Simile uses EPO techniques.

Replication Routine

Simile contains a fairly basic direct action replication mechanism that attacks PE files on the local machine and the network. The emphasis is clearly on the metamorphic engine, which is unusually complex.

EPO Mechanism

The virus searches and replaces all of the possible patterns of certain call instructions (those that reference ExitProcess() API calls) to point to the beginning of the virus code. Thus the main entry point of the file is not altered.

Sometimes the metamorphic virus body is placed together with a polymorphic decryptor at the same location within the file. In other cases the polymorphic decryptor is placed at the end of the code section, while the virus body is placed in another section. This is to conceal further the location of the virus body.

Polymorphic Decryptor

During the execution of an infected program, when the instruction flow reaches one of the hooks that the virus has placed in the code section, control is transferred to a polymorphic decryptor which is responsible for decoding the virus body (or simply copying it directly since, intentionally, the virus body is not always encrypted.)

This decryptor, whose location in the file is variable, allocates a large chunk of memory (about 3.5 megabytes) then proceeds to decipher the encrypted body into it. It does this in a most unusual manner: rather than going through the encrypted data linearly, it processes it in a seemingly random order, thus managing to avoid triggering some decryption-loop recognition heuristics.

This 'Pseudo-Random Index Decryption', as the virus writer calls it, relies on the use of a family of functions that have interesting arithmetic properties, modulo 2^n.

While the virus writer discovered this by a process of trial and error, it is possible to produce a mathematical proof that his algorithm works in all cases (provided the implementation is correct, of course). Such a proof is beyond the scope of this article but the proof, by Frédéric Perriot, is available at http://www.peterszor.com/.

The size and appearance of the decryptor varies greatly from one virus sample to the next. To achieve this high level of variability, the virus writer simply generates a code template and then puts his metamorphic engine to work to transform the template into a working decryptor!

In some cases, the decryptor may start with a header whose intent is not immediately obvious upon reading it. Further study reveals that its purpose is to generate anti-emulation code on the fly: the virus constructs a small oligomorphic code snippet containing the instruction RDTSC ('ReaD Time Stamp Counter'). This retrieves the current value of an internal processor ticks counter. Then, based on one random bit of this value, the decryptor either decodes and executes the virus body or bypasses the decryption logic altogether and simply exits.

Besides confusing emulators that do not support the somewhat peculiar RDTSC instruction (one of The Mental Driller's favourites, which he used previously in W95/Drill), this is also a very strong attack against all algorithms that rely on emulation either to decrypt the virus body or to determine viral behaviour heuristically. Effectively, it causes some virus samples to cease infecting completely upon a random time condition.

On initial execution, the virus body will retrieve the addresses of 20 APIs that it requires for replication and for displaying the payload.

Next the virus will check the system date in order to determine whether either of its payloads should activate. Both payloads require that the host imports functions from User32.dll. In this case, the virus checks whether it should call the payload routine or not (which is explained below).

Metamorphism

After the payload check has completed, a new virus body is generated. This code generation is carried out in a number of steps:

The first step is to disassemble the viral code into an intermediate form, which is independent of the CPU upon which the native code will execute. This allows for future extensions, such as producing code for different operating systems or even different CPUs.

The second step is to shrink the intermediate form, by removing the redundant and unused instructions. These instructions were added by earlier replications to interfere with disassembly by virus researchers.

The third step is to permutate the intermediate form, for example reordering subroutines, or separating blocks of code and linking them with jump instructions.

The fourth step is to expand the code, by adding redundant and unused instructions.

The fifth step is to reassemble the intermediate form into a final native form that will be added to infected files.

Thus Simile can not only expand, as most first generation metamorphic viruses do, but it can also shrink (and shrink to different forms!).

Replication

Next the replication phase begins. It starts by searching for *.exe in the current directory, then on all fixed and mapped network drives.

The infection will scan recursively into directories, but only to a depth of three subdirectories, and avoiding completely any directory that begins with the letter 'W'.

For each file that is found, there is a 50% chance that it will be skipped explicitly. Additionally, files will be skipped if they begin with 'F-', 'PA', 'SC', 'DR', 'NO', or contain the letter 'V' anywhere in the name.

Due to the nature of the comparison, other character combinations are skipped unintentionally, for example any directory that begins with the number 7, any file that begins with 'FM', or any file that contains the number 6 anywhere in its name.

The file infection routine contains many checks to filter files that cannot be infected safely. For example, the file must contain a checksum, it must be an executable for the Intel 386+ platform, and there must exist sections whose names are '.text' or 'CODE', and '.data' or 'DATA'. The virus also checks that the host imports some kernel functions, such as 'ExitProcess'.

For any file that is considered infectable, random factors and the file structure will determine where the virus places its decryptor and virus body.

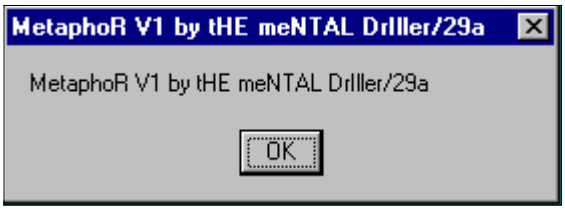
If the file contains no relocations, or with only a small chance, the virus body will be appended to the last section in the file. In this case, the decryptor will be placed either immediately before the virus body, or at the end of the code section.

Otherwise, if the name of the last section is '.reloc', the virus will insert itself at the beginning of the data section and move all of the following data and update all of the offsets in the file.

Payload

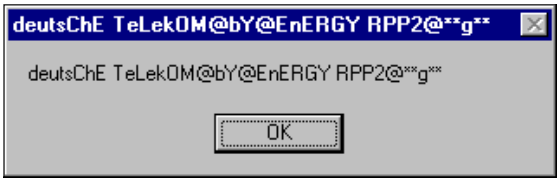
The first payload activates only during the months of March, June, September, and December. Variants A and B of W32/Simile display their message on the 17th day of these months. Variant C will display its message on the 18th day of these months.

Variant A will display the message 'Metaphor v1 by The Mental Driller/29A':



and variant B will display 'Metaphor 1b by The Mental Driller/29A':

Variant C attempts to display 'Deutsche Telekom by Energy 2002 **g**':



However the author of variant C had little understanding of the code, and the message rarely appears correctly. In all variants, the message appears in randomly mixed letter cases.

The second payload activates on 14 May in variants A and B, and on 14 July in variant C.

In the second payload, variants A and B will display the message 'Free Palestine!' on computers that use the Hebrew locale. Variant C attempts to display the text 'Heavy Good Code!' but, due to a bug in the virus code, this message is displayed only on systems on which the locale cannot be determined.

Conclusion

During the extensive and detailed tests carried out with W32/Simile replication on test systems we have noticed that the virus code generates garbage unintentionally or trashes some files accidentally as the direct result of its extreme complexity.

It seems that obfuscated code is not only challenging for virus researchers to analyse, but it is very challenging for the author of the code to debug.

As the saying goes: 'there are three kinds of lies: lies, damn lies, and statistics'. The complex infection mechanism coupled with the powerful metamorphic engine make it very difficult to reach 100% accuracy using only empirical evaluation methods, and indepth analysis of the virus code is essential.

Exact identification becomes a problem even for humans. How long does it take to be sure if something is really variant A or C or a new one? Is it modified or is it the same? It is becoming more difficult to know. The need to understand metamorphic code in a quicker fashion must be the subject of further research.

As this issue of *VB* goes to print, W32/Simile (aka W32/Etap) appears on the preliminary April 2002 supplemental WildList.

W32/Simile

Alias:	W32.Etap, Metaphor.
Type:	Direct action Win32, portable executable infector, complete metamorphic virus.
Removal:	Detect and delete infected files and replace them from clean backups.
Payload:	Displays messages on certain dates.
Unintended payload:	Trashes some portable executable files.