

An Analysis of Simile

Adrian Marinescu 2003-03-04

An Analysis of Simile

by [Adrian Marinescu](#)

last updated March 4, 2003

Virus writers have always tried to develop new methods to make malware detection more difficult. For instance, encryption was a natural step in virus evolution when scanners started to use databases with scan strings for detection. When scanners started to handle encryption patterns generically, first oligomorphism (a limited form of polymorphism - the polymorphic decryptor can have a strictly limited, relatively small number of shapes) and then polymorphism were introduced. Then, as emulation was used more and more by antivirus programs, it became clear that new methods must be developed to hide the viral code. For example, [Ply](#) was a simple DOS virus that used an interesting technique, based on the fact that Intel opcodes are variable in size. It padded every instruction that was not 3 bytes in length with no-operation instructions. These 3-byte pieces would be randomly linked with jumps, and, inside each piece that contained a non-3 byte opcode, the padded instruction(s) could be shifted. In 1998, another virus appeared that was able to show something new. [Lexotran](#) was a polymorphic DOS virus; but the polymorphism was unusual: there were no constant parts of the code, even after decryption. Another DOS virus, [ACG](#), which was developed even earlier, was one of the first viruses that had no constant parts in the body.

In the beginning, polymorphic viruses had a variable decryptor generated by the polymorphic engine that decrypted and gave control to the virus body, which was constant after decryption. To generically handle such viruses, antivirus engines emulated the decryptor and then the same methods used to detect plain viruses could be used. Lexotran and ACG tried to avoid this by using a rudimentary mechanism that would allow only small parts of the code to be constant. A "skeleton" was kept in encrypted or compressed form and this was used to generate new shapes of the virus. To detect this, scanners had to emulate everything including the infection process. The constant parts were only present when a file was infected. The idea behind this was very simple - one simple instruction can be represented in a lot of different forms. For instance, a "1" value in a register can be stored either by storing it directly or by generating a lot of mathematical operations that will result in a "1" value and then storing it. Moreover, storing a variable can be done multiple ways. For instance, one or more temporary variables could be used.

This method was catalogued by antivirus researchers as "body-polymorphism" or "metamorphism". It was very clear that more and more viruses based on this mechanism were going to be written, and, more importantly, that the method would be used on other platforms, too. Viruses like [Win95/Zmist](#) and [Win95/Puron](#) were simply based on the same techniques that ZCME and Lexotran had used in the past.

At the same time, other methods were developed to avoid detection. Entry point obscuring (EPO) was just one of them. The idea behind EPO was to avoid calling the virus directly when an infected program runs, and to instead analyze the program and find a place to insert the virus code. This way, an antivirus scanner would be forced to analyze a lot of data that is normally skipped, resulting in a scanner slowdown. Another method was permutation, which was the ability of viruses to split themselves into pieces that could be permuted when a new shape is generated. The [BadBoy DOS](#) virus family was the first one to use this technique. On Windows, [Win95/Smash](#) was one of the first that successfully implemented permutation. Finally, virus writers used the cross-platform infection to increase the number of available infection vectors, which would therefore increase the chances of viruses to spread.

The [Win32/Linux Lindose.2132.A](#) the first one that was able to infect both Windows and Linux executables from both the operating environments.

Thus, it was no surprise when a virus that combined these methods was discovered. Called [Win32/Simile](#), the virus had no constant parts that could be used for detection. It used EPO by searching for calls to the "ExitProcess" API and replacing them with calls to the virus entry point. Also, several infection methods were used that would make file structure less suspicious for heuristic analyzers, such as mid-infection, the ability to insert the virus code in the middle of the file, not at the beginning or at the end like most of the viruses are doing.

In a short period after the original variant was released, two other variants were also discovered. No major differences were encountered, so both were probably recompilations based on the original source code. The fourth variant discovered had a lot of parts in common with the first three, but was also able to infect Linux/i386 ELF executable files. In fact, Simile.D is the first polymorphic and metamorphic cross-platform infector and is, by far, the most complex multipartite virus discovered until now.

Infection

The infection process is fairly simple: Simile searches for files and, depending on the inner file format, attempts to infect them. If the file is a PE executable file, several checks will be made. It avoids infecting files that match several known antivirus utilities, as well as files located in folders with "W" as the first letter of the name, this way it will avoid infecting files from the Windows folder. Depending on the internal PE executable file layout, one of three possible methods of infection is selected: either the decryptor will be located in the code section and the virus body will be in the data section, the virus body will be in the last section and the decryptor will be in the code section, or, finally, the virus body and the decryptor will both be in the last section.

Simile is also an entry-point obscuring virus. If the file to be infected imports `ExitProcess` or `_exit` APIs (from `KERNEL32.DLL` or `MSVCRT.DLL`) it will patch the API call with a call to the virus decryptor. This method was used in the past to make detection harder and slow down the scanning process, but was never combined with a metamorphic approach. The ELF executable infection is pretty much a port of the Win32 algorithm; ELF files are infected by using a single method of infection. Calls to `exit` are replaced with calls to the virus decryptor. The decryptor and the virus body will be placed in a newly created section and the section will be inserted with a randomly generated name. Unlike the first known virus to infect both Windows and Linux executables (`{Win32,Linux}/Lindose.2132.A`) that had completely different parts to infect each file type, Simile shares a lot of the code between the platforms.

As has been seen in previous viruses written by the same author (such as [Win95/Driller](#), which uses the Tuareg polymorphic engine), the encryption algorithm is not linear, and it's based on modulo arithmetic. There is a random 1-in-16 probability that the decryptor will just copy the virus, so the virus will not be encrypted. To avoid detection based on emulation, with a probability of 1-in-4 decryptors, Simile will randomly exit or continue execution.

Infected files will grow in size up to around 100Kb, but the size might vary a lot, as in some cases in which the virus can expand, and, other cases in which it might shrink.

Metamorphism

Traditionally, viruses that did not use some kind of P-CODE to generate a new shape had a big disadvantage: when creating a new shape, the generated code could only grow in size. [Win32.Evol](#) is one of them. After less than ten generations the virus body grows to about 1Mb, at which point the infection process becomes pretty slow and buggy.

Simile found a way to avoid this problem: from generation to generation the virus is able to both grow and shrink, depending on some random counters. To do this, Simile uses an algorithm that is based on dynamic code translation techniques, as well as "just in time" compilation. First of all, it disassembles itself into an internal assembler that supports all the basic opcodes. All further operations are performed using this internal P-CODE, which results in another interesting feature that might be implemented in the future by the virus author, Cross-platform on different architectures, not only on Intel processors. The disassembled code is shrunk by using several patterns for reduction, all of which are generated by the code expander. For instance, a sequence of arithmetic operations with immediate constants is reduced to a single instruction with the result of the other operations. After the code is reduced, it will be split into chunks, which will be permuted and then linked. On completion, an expansion algorithm will be performed. For instance, a memory load operation can be done directly or by using one or more temporary registers and/or memory locations. The expanded code is compiled into native code and the result will be the new shape of the virus.

Of course, such complex mechanisms will require resources; Simile's metamorphic engine requires about 3 1/2 megabytes of memory and is by far the biggest engine used in a virus. Even though the rest of the virus does not require as much memory as Win32/Zmist does, (to decompile the host program Zmist requires at least 32 Mb of memory) on systems with very low resources the infection will be easily noticed.

Interestingly enough, some of the metamorphic engine features are only used with a very low probability. This way the work of antivirus program is made harder because these features will result in infected files that will be completely different from the common ones.

Detection Issues

Simile has no harmful payload; it only displays message boxes on Windows on several days/months and only prints a message when running on Linux. Despite this, detection of such viruses is very important. One of the main concerns is that another, far more dangerous, virus could use the metamorphic engine very easily. In such a case, it would simply not be acceptable to have detection made available only a long time after the virus was discovered. Besides the complexity and the time needed to detect such viruses, antivirus researchers face another problem. Developing such routines usually results in a scanner slowdown, and sometimes in false positives and/or negatives. That's why developing generic solutions for such

complex viruses is an interesting area of research for the community these days.

Final Words

Even if Simile virus family does not represent any new ideas, the complexity of the metamorphic engine makes it unique. Despite its complexity, Simile is stable enough to pass by unnoticed in an operating environment. Even though the cross-platform infection is not supposed to increase the chances of spreading, another malware unintentionally carrying Simile might result in trouble in the future. Also, having the source code for such a complex virus available publicly on the Internet is another problem that might help other virus authors to develop similar techniques in the future. It is very clear that in the same way that emulators were introduced to handle the growing number of polymorphic viruses, some generic approach will need to be developed to handle complex creations such as Simile faster and easier.

Simile Vital Statistics

Virus name: {Win32,Linux}/Simile.D

Aliases: Win32/Etap.D, Linux/Etap.D, Win32/Etapux, Etapux

Type: Direct action infector running on both Windows and Linux

Infected targets: PE executable files and ELF executable files

Size: Variable, no less than 64Kb, usually around 110Kb

Removal: Use antivirus software to detect and delete the infected files, restore the files from backup.

[Privacy Statement](#)

Copyright 2006, SecurityFocus