Metamorphic Virus: Analysis and Detection

This thesis describes the evolution of the first simple computer virus to the most advanced metamorphic virus.

BY EVGENIOS KONSTANTINOU AND STEPHEN WOLTHUSEN

Royal Holloway University of London

ABSTRACT

Metamorphic viruses transform their code as they propagate, thus evading detection by static signature-based virus scanners, while keeping their functionality. They use code obfuscation techniques to challenge deeper static analysis and can also beat dynamic analyzers, such as emulators, by altering their behavior. To achieve this, metamorphic viruses use several metamorphic transformations, including register renaming, code permutation, code expansion, code shrinking, and garbage code insertion. In this article, a simple analysis of metamorphic viruses is presented, along with the techniques they use to transform their code to new generations. This article describes the evolution of the computer virus from the first-generation simple virus to the most advanced metamorphic virus. Several metamorphic techniques are described, then the description of several techniques to detect metamorphic viruses is given.

Evgenios Konstantinou

Information Security Group, Royal Holloway, Egham, Surrey, U.K.

Stephen Wolthusen

Information Security Group, Royal Holloway, Egham, Surrey, U.K.

Metamorphic Virus: Analysis and Detection

1 THE MALWARE MENACE

The recent years have been very interesting, but at the same time very frustrating for the information security professional. As information technology is expanding and improving, so are its threats. Its adversaries evolved from the 15 year old "script kiddy" to the professional hacker employed by organized crime.

A recent research in the UK showed that around 97% of businesses in the UK have internet connection and around 88% have broadband, thus the thread from malicious software has never been greater[1].

The research reports that virus infection was the biggest single cause of respondents' worst security incidents, accounting for roughly half of them. Two-fifths of these were described as having a serious business impact. The report also informs that virus infections tended to take more effort to resolve than other incidents, some of

This article was prepared by students and staff involved with the award-winning M.Sc. in Information Security offered by the Information Security Group at Royal Holloway, University of London. The student was judged to have produced an outstanding M.Sc. thesis on a business-related topic. The full thesis is available as a technical report on the Royal Holloway website http://www.ma.rhul.ac.uk/tech.

For more information about the Information Security Group at Royal Holloway or on the M.Sc. in Information Security, please visit http://www.isg.rhul.ac.uk.

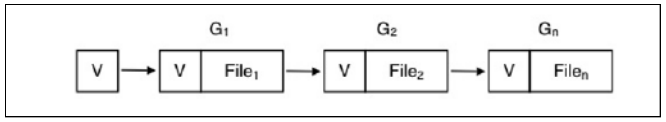
them needing more than 50 person-days.

With the exception of rootkits, metamorphic viruses must be the most sophisticated malicious pieces of code. To write a decent metamorphic engine is a very challenging task and some of them are so well written that modern antivirus products can still miss them some times, as shown by Christodorescu and Jha in[2].

Because of their complexity their study is very interesting, and the fact that there were no real metamorphic viruses in the wild since Simile in 2002 should not make the virus researcher relaxed. The technology is there and waiting to be exploited and implemented into modern types of malware, such as network worms and spyware.

Computer Virus. A computer virus is a malicious program that modifies other host files or boot areas to replicate. In most cases the host object is modified to include a complete copy of the malicious code program. The subsequent running of the infected host file or boot area then infects other objects[3]. There are many types of viruses such as Boot Sector viruses, File Infecting viruses, Memory Resistant viruses, Macro viruses, etc. Figure 1 illustrates the simple virus V replication, from generation to generation.

Figure 1: Simple virus replication



2 VIRUS DETECTION MECHANISMS

Before digging into more advanced computer viruses, a description of the most widely used detection techniques is appropriate.

2.1 String Scanning

String scanning is the simplest technique used by anti-virus software to detect computer viruses. It searches for sequence of bytes (strings) that are typical of a specific virus but not likely to be found in other programs. This sequence of bytes is often called the signature of the virus[4].

2.2 Wildcards

Scanners that support wildcards are allowed to skip bytes or byte ranges. For example, the bytes represented by the '?' character are skipped. Some early generation encrypted, polymorphic, and even metamorphic viruses could be detected using wildcards.

A computer virus is a malicious program that modifies other host files or boot areas to replicate.

2.3 Heuristics Analysis

Heuristics analysis is useful when detecting new viruses. This technique is also particularly useful for detecting macro viruses. For binary viruses heuristic analysis can be helpful, but it creates many false positives which is a big problem. However, in many cases a heuristic analyzer can be valuable and can also be used to detect variants of existing virus families.

Heuristic analysis, as described in[5], can be static or dynamic. Static analysis base the analysis on file format and the code structure of the virus body. Dynamic heuristics use emulators to detect suspicious behavior while the virus code runs inside the emulator.

2.4 Algorithmic Scanning

There are cases when the standard algorithm of the virus scanner cannot deal with a virus and a new detection code must be introduced to implement a virus-specific detection algorithm. This method is called algorithmic scanning. Early scanners implemented algorithmic scanning by hard-coding detection routines that were released with the core engine code, but this technique caused many problems like stability issues of the scanner. To solve this problem, researchers introduced virus scanning languages, which allowed seek and read operations in scanned objects[4].

2.5 Code Emulation

This extremely powerful technique implements a virtual machine to simulate the CPU and memory management system and executes malicious code inside the virtual machine. The malicious code cannot escape the virtual machine of the scanner, thus this technique is relatively safe. The code emulator mimics the instruction set of the CPU using virtual registers and flags. The functionality of the operating system must also be emulated to create a virtualized system that supports system APIs, files, memory management, etc. The emulator mimics the execution of programs and analyzes each instruction one-by-one.

3 ADVANCED CODE EVOLUTION

Malware writers are continually trying to invent new methods to defeat antivirus software. Their worst enemies are the most commercially popular antivirus products. Virus writers had to come up with ideas that made first-generation virus scanners useless.

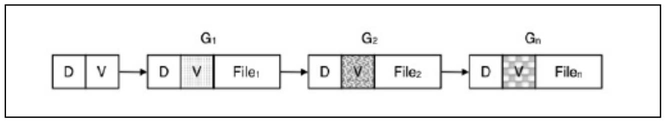
Malware writers are continually trying to invent new methods to defeat antivirus software.

3.1 Encrypted Virus
One of the first and easiest methods virus writers used to hide the functionality of the virus code was encryption. Usually, an encrypted virus consists of two parts; the decryptor and the encrypted main body of the virus. The decryptor executes when an infected program runs, and decrypts the virus body. In[6] it is mentioned that virus writers use encryption for the four following reasons:

- prevent static code analysis
- prolong the process of dissection
- prevent tampering
- evade detection

Figure 2 illustrates how the encrypted virus replicates. The decryptor D is constant and behind the encryption the body of the virus remains constant too.

Figure 2: Encrypted virus replication

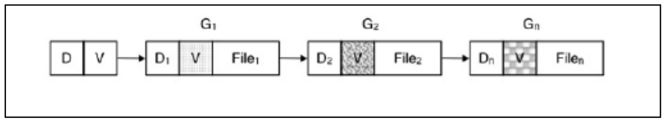


3.2 Polymorphic Virus
Polymorphism is the next step virus writers took to challenge antivirus software. Polymorphic viruses can mutate their decryptors to a high number of different instances that take millions of different forms[4]. They use their mutation engine to create a new decryption routine each time they infect a program. The new decryption routine would have exactly the same functionality, but the sequence of instructions could be completely different[7].

The mutation engine also generates an encryption routine to encrypt the static code of the virus before it infects a new file. Then the virus appends the new decryption routine together with the encrypted virus body onto the targeted file. Since the virus body is encrypted and the decryption routine is different for each infection, antivirus scanners cannot detect the virus by using search strings[7].

Figure 3 illustrates how the polymorphic virus replicates. The decryptor D changes shape from generation to generation, but behind the encryption there is still a constant virus body.

Figure 3: Polymorphic virus replication



The mutation engine also generates an encryption routine to encrypt the static code of the virus before it infects a new file.

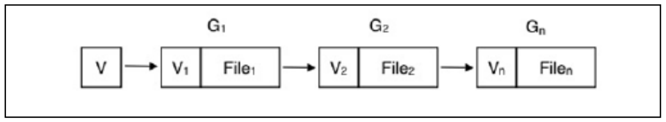
4 METAMORPHIC VIRUSES

Metamorphic viruses transform their code as they propagate, thus evading detection by static signature-based virus scanners and have the potential to lead to a breed of malicious programs that are virtually undetectable statistically[8]. These viruses also use code obfuscation techniques to challenge deeper static analysis and can also beat dynamic analyzers, such as emulators, by altering their behaviour when they detect that they are executing under a controlled environment[9].

Metamorphic viruses do not have a decryptor and do not "unpack" to give a constant virus body like polymorphic viruses do. However, they are able to create new generations of the virus that look different. They do not use a data area filled with string constants but have one single-code body that carries data as code[4]. To achieve this, metamorphic viruses use several metamorphic transformations, such as register usage exchange, code permutation, code expansion, code shrinking, and garbage code insertion.

Figure 4 illustrates the replication of a metamorphic virus. It is obvious that no constant data exists between different generations.

Figure 4: Metamorphic virus replication



4.1 Metamorphic Techniques

To avoid detection, metamorphic viruses use several different techniques to evolve their code into new generations that look completely different, but have exactly the same functionality.

4.1.1 Garbage Code Insertion

Garbage code (or junk code) insertion is a simple technique used by many metamorphic and polymorphic viruses to evolve their code. The idea behind this technique is to make their code look different so that no usable hexadecimal search string can be extracted. The instructions inserted into the code are called garbage because they have no impact on the functionality of the code[10].

4.1.2 Register usage exchange

Another simple technique used by metamorphic viruses is register usage exchange. This

Metamorphic viruses do not have a decryptor and do not "unpack" to give a constant virus body like polymorphic viruses do.

method was used by the Win95/Regswap virus, which was created by the virus writer Vecna and released in 1998. Different generations of the virus will use the same code but with different registers[11].

4.1.3 Permutation Techniques

The Win32/Ghost and the Win95/Zperm viruses introduced a new level of metamorphism. Although the virus code is constant, metamorphosis is achieved by dividing the code into frames, and then position the frames randomly and connect them by branch instructions to maintain the process flow. The flow of control always remains the same[12].

The Win32/Ghost virus, which was discovered in May 2000, had the ability to re-order its subroutines from generation to generation. If the number of subroutines is n, then the number of different virus generations is n!. Win32/Ghost had 10 subroutines, thus there were 3628800 different possible virus generations.

4.1.4 Insertion of Jump Instructions

Another method used by some metamorphic viruses to create new generations is inserting jump instructions within its code. The Win95/Zperm virus is a very good example of this technique. The virus inserts and removes jump instructions within its code and each jump instruction will point to a new instruction of the virus[11]. Zperm never generates a constant body anywhere, not even in memory, so detection of the virus using search strings is virtually impossible.

4.1.5 Instruction Replacement

Some metamorphic viruses are able to replace some of their instructions with other equivalent instructions. In addition to jump insertions, Win95/Zperm had the ability to perform instruction replacement. For example, the virus could replace the instruction "xor eax, ea" with the instruction "sub eax, eax". Both instructions perform the same function – zeroing the content of the eax register – but have a different opcode (hexadecimal representation of the instruction)[11].

Another example of instruction replacement is the Win95/Zmist virus.

The types of instruction replacement that can be performed by Zmist, as described in [13], include:

- reversing of branch conditions
- register moves replaced by push/pop sequences

Another method used by some metamorphic viruses to create new generations is inserting jump instructions within its code.

- alternative opcode encoding
- xor/sub and or/test interchanging

4.1.6 Host Code Mutation

The Win95/Bistro virus not only mutates itself in new generations, but it also mutates the code of its host. This way the virus can generate new viruses and worms. To do this, the virus uses a randomly executed code-morphing routine. Also, because the entry-point code of the application could be different, disinfection cannot be done perfectly.

The code-morphing routine of Bistro uses techniques previously described in this section. Code permutations of worms and viruses, as done by Bistro, would be very difficult to deal with. If similar morphing techniques were implemented by a 32-bit worm, a major problem would occur as new mutations of old viruses and worms would be created endlessly[10].

4.1.7 Code Integration

The Win95/Zmist virus implemented an even more sophisticated technique. This technique, named code integration, has never been seen in any previous virus. Zmists engine can decompile Portable Executable (PE) files to their smallest elements, requiring 32MB of memory. Then the virus moves code blocks out of the way, inserts itself into the code, re-generates code and data references, and rebuilds the executable[11]. This way the virus can integrate itself seamlessly to the code of its target, making it very hard to detect and even harder to repair.

4.2 Advanced Metamorphic Viruses

Win95/Zmist and Win32, Linux/Simile were the two most advanced metamorphic viruses. Zmist was created by the virus writer Z0mbie and released in 2000. Simile – named “MetaPHOR” by its creator – was created by “The Mental Driller” and was released in 2002.

4.2.1 Win95/Zmist

The Russian virus writer Z0mbie released Win95/Zmist in 2000, along with his “Total Zombification” magazine. Z0mbie is the author of many other polymorphic and metamorphic viruses, including Win95/Zmorph and Win95/Zperm. At the time of its release, Zmist was one of the most complex viruses. Peter Ferrie and Peter Szor went as far as to call Zmist “one of the most complex binary viruses ever written.” Zmist is a Entry-Point Obscuring (EPO) virus that is metamorphic. With the EPO method, some random place in the victims’ body is patched by virus

Code permutations of worms and viruses, as done by Bistro, would be very difficult to deal with.

instructions in the hope that these instructions will gain control at some point[14].

In addition, Zmist randomly uses a polymorphic decryptor[13].

Zmist supports the unique technique called code integration. Also, it occasionally inserts jump instructions after every single instruction of the code section, each pointing to the next instruction. The fact that these extremely modified applications work – from generation to generation – was not expected by anyone, not even by Z0mbie. In [13] it is mentioned that "due to its extreme camouflage, Zmist is clearly the perfect anti-heuristics virus."

4.2.2 Win32, Linux/Simile

In March 2002, a virus writer who calls himself "The Mental Driller," released the Win32/Simile virus. Information about Simile comes from[15].

Simile, which is even more complex than Zmist, is approximately 14,000 lines of assembly code. Its extremely powerful and complex metamorphic engine takes up about 90% of the virus code. His creator named the virus "MetaPHOR", which stands for Metaphoric Permutating High-Obfuscating Reassembler.

There are four known variants of the virus, three of them (variants A, B, and D) written by the original author, and one (variant C) written by an unknown author[4].

Simile is very obfuscated and very difficult to understand. It attacks the disassembling, debugging, and emulation techniques. It also challenges the standard evaluation-based techniques for virus analysis. Just like Zmist, Simile makes use of EPO techniques. Most first generation metamorphic viruses could only expand. Simile can both expand and shrink to different forms. The power of Similes' engine is demonstrated in the following code, which was published in[16]:

```
mov dword_1, 0h
mov edx, dword_1
mov dword_2, edx
mov ebx, dword_2
mov edi, 32336C65h
lea eax, [edi]
mov esi, 0A624548h
or esi, 4670214Bh
lea edi, [eax]
mov dword_4, edi
mov edx, ebp
mov dword_5, edx
mov dword_3, esi
mov edx, offset dword_3
```

Simile is very obfuscated and very difficult to understand. It attacks the disassembling, debugging, and emulation techniques.

```
push edx
mov dword_6, offset GetModuleHandleA
push dword_6
pop dword_7
mov edx, dword_7
call dword ptr ds:0[edx]
```

Similes' metamorphic engine could replace the previous code by the following five lines:

```
mov dword_3, 6E72654Bh
mov dword_4, 32336C65h
mov dword_5, 0h
push offset dword_3
call ds:[GetModuleHandleA]
```

5 METAMORPHIC VIRUS DETECTION

Metamorphic techniques make virus detection using search strings virtually impossible. To detect a metamorphic virus, techniques such as examination of the file structure, or analysis of the behavior of the code must be used. For perfect detection of a metamorphic virus, detection routines must be written that can generate the essential instruction set of the virus body from the actual instance of the infection[4].

5.1 Geometric Detection

Geometric detection is based on modifications that a virus has made to the file structure. Peter Szor calls this method shape heuristics because it is far from exact and prone to false positives[4]. Geometric detection can be used to detect Win95/Zmist. Because the data section of a file is increased by at least 32KB when it is infected by an encrypted version of the virus, the file might be reported as being infected if the virtual size of its data section is at least 32KB larger than its physical size. However, this method could introduce false positives[4].

5.2 Wildcard String and Half-Byte Scanning

Simple metamorphic viruses, such as viruses that use register swapping and instruction replacement, can be detected by wildcard and half-byte scanning. For example, in the Win95/Regswap virus that there exist many common opcodes that are constant to all generations of the virus. This makes the extraction of usable search strings using wildcards possible. If the scanner supports it, half-byte detection would also be appropriate for this type of infection[4].

Metamorphic techniques make virus detection using search strings virtually impossible.

5.3 Code Disassembling

Disassembling the virus code means separating the stream into individual instructions. This technique is good for detecting viruses that insert garbage code between their code. Code disassembling becomes a powerful tool when combined with a state machine, which could record the order in which "interesting" instructions are found. (A state machine is a model of behaviour composed of a finite number of states, transitions between those states, and actions [17]) It becomes even more powerful if it is combined with an emulator, and it becomes capable of detecting difficult viruses like Win95/Zmist or Win95/Puron based on an engine called "Lexotan"[4].

5.4 Using Emulators

Code emulation implements a virtual machine to simulate the CPU and memory management system and executes malicious code inside the virtual machine. The malicious code cannot escape the virtual machine of the scanner[4]. Antivirus scanners can run code inside an emulator and examine it periodically or when interesting instructions are executed.

5.4.1 Using Emulator-Based Heuristics

Heuristic detection does not identify viruses specifically but extracts features of viruses and detects classes of computer viruses generically. The emulator-based heuristics technique is described in[4].

The heuristics engine can track the interrupts or implement a deeper level of heuristics using a virtual machine that simulates the operating system.

Such systems can even replicate the virus inside the virtual machine on a virtual file system. Some antivirus products implement such systems and find them to be very effective, providing less false positives. This technique requires emulation of file systems. For example, whenever a new file is opened by the emulated virus, a virtual file is given to it. Then the emulated virus might decide to infect the virtual file in its own virtual system. The two biggest problems is that is very difficult to emulate multithreaded systems and performance is poor.

5.4.2 Dummy Loops Detection

An anti-emulation technique was introduced by an improved version of the Bistro virus, which was released some time after the original. This technique, which is called random

Heuristic detection does not identify viruses specifically but extracts features of viruses and detects classes of computer viruses generically.

code insertion, inserts garbage instructions and dummy loops randomly before the decryptor code. This forces some emulators to emulate millions of garbage instructions and fail to rebuild the real virus. This results in failure to detect the virus[12].

5.4.3 Stack Decryption Detection

Variants of the Zmorph virus place a piece of polymorphic code at the entry point of an infected file. Then they decrypt the virus instruction-by-instruction and rebuild it by pushing the result into the stack memory. If the emulator is not capable of detecting stack decryption, such viruses would be missed. The memory accessed by the virus must be monitored by the emulator and when control is transferred to the stack memory, the emulator should detect it and dump the whole decrypted virus code for identification.

The drawback of this technique is that is has a significant impact on the performance of the scanner[12].

5.5 Code Transformation Detection

Code transformation is used to convert mutated instructions into their simplest form, where the combinations of instructions are transformed to an equivalent but simple form. After the transformation, common code exhibited by the virus can be identified[12]. The first metamorphic virus that this technique was applicable to was Win32/Simile. This technique involves transforming the virus code back to its initial form similar to the first generation. However, to be able to guarantee perfect detection without compromising scanning speed, the code transformation module must be highly optimised and flexible. The virus location can be transformed to where the scan pattern is taken this will reduce the impact on the performance of the scanner[12].

5.6 Subroutine Depermutation

Subroutine depermutation technique is used for detection of viruses that use permutation of their code to form new generations. As described earlier, metamorphosis is achieved by dividing the code into frames, and then positioning the frames randomly and connecting them by branch instructions to maintain the process flow[12].

The Zperm virus uses the sophisticated Real Permutation Engine (RPME) in order to mutate its code. To detect such a virus, the scanner must perform partial emulation to

Variants of the Zmorph virus place a piece of polymorphic code at the entry point of an infected file.

reconstruct the virus code into its initial form before the permutation. Partial emulation means emulating branch instructions, such as jump instructions. Deciding when to stop decoding is the problem of this technique. Also, ensuring that the virus code is finished is another challenge. In addition to rebuilding the virus code, this technique can be effective for removing garbage instructions too[12].

6 CONCLUSION

This article described the evolution of the computer virus, from the first-generation simple virus to the most advanced metamorphic virus. First-generation viruses are simple and their detection is relatively trivial. As anti-virus software became more advanced, virus writers kept inventing new techniques to thwart detection. This lead to the creation of the metamorphic virus, which used advanced code mutation techniques.

This article described many techniques used by metamorphic viruses, such as Garbage Code Insertion, Instruction Replacement, Host code Mutation, and Code Integration.

Anti-virus vendors had to react and invent several detection techniques, such as Geometric Detection, Code Disassembling, Stack Description Detection, and Subroutine Depermutation, capable of detecting metamorphic viruses.

However, some metamorphic viruses are so advanced that anti-virus software are not able to detect them even today. Fortunately, these viruses are so difficult to write that virus writers turned to other types of malware. This, however, does not mean anti-viruses researchers can relax as viruses writers are beginning to use metamorphic techniques again, in different types of malware.*

Ron Condon

UK bureau chief
searchsecurity.co.UK

Ron Condon has been writing about developments in the IT industry for more than 30 years. In that time, he has charted the evolution from big mainframes, to minicomputers and PCs in the 1980s, and the rise of the Internet over the last decade or so. In recent years he has specialized in information security. He has edited daily, weekly and monthly publications, and has written for national and regional newspapers, in Europe and the U.S.

REFERENCES

[1] Alun Michael, Chris Potter, and Andrew Beard. Information security breaches survey 2006. Technical report, PriceWaterhouseCoopers, 2006.

[2] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In SSYM'03: Proceed-

ings of the 12th conference on USENIX Security Symposium, pages 12–12, Berkeley, CA, USA, 2003. USENIX Association.

[3] Roger A. Grimes. Malicious Mobile Code: Virus Protection for Windows. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[4] Peter Szor. The Art of Computer Virus Research and Defense. Addison Wesley Professional, 1 edition, February 2005. 12

[5] Prabhat K. Singh and Arun Lakhotia. Analysis and detection of computer viruses and worms: an annotated bibliography. SIGPLAN Not., 37(2):29–35, 2002.

[6] Fridrik Skulason. Virus encryption techniques. Virus Bul letin, pages 13–16, November 1990.

[7] Carey Nachenberg. Computer virus-antivirus coevolution. Commun. ACM, 40(1):46–51, 1997.

[8] Mohamed R. Chouchane and Arun Lakhotia. Using engine signature to detect metamorphic malware. In WORM '06: Proceedings of the 4th ACM workshop on Recurring malcode, pages 73–78, New York, NY, USA, 2006. ACM Press.

[9] Arun Lakhotia, Aditya Kapoor, and Eric Uday Kumar. Are metamorphic computer viruses really invisible? part 1. Virus Bul letin, pages 5–7, December 2004.

[10] Peter Szor. The new 32-bit medusa. Virus Bul letin, pages 8–10, December 2000.

[11] Peter Sz ?or and Peter Ferrie. Hunting for metamorphic. In Virus Bulletin Conference, September 2001.

[12] Rodelio G. Finones and Richard t. Fernandez. Solving the metamorphic puzzle. Virus Bul letin, pages 14–19, March 2006.

[13] Peter Ferrie and Peter Szor. Zmist oportunities. Virus Bul letin, pages 6–7, March 2001.

[14] Malivanchuk Taras. Epo - what is next? Virus Bul letin, pages 8–9, March 2002.

[15] Frederic Perriot, Peter Szor, and Peter Ferrie. Striking similarites: Win32/simile and metamorphic virus code. Technical report, Symantec, 2003.

[16] Myles Jordan. Dealing with metamorphism. Virus Bul letin, pages 4–6, Octomber 2002.

[17] Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, and Peter Wolstenholme. Modeling Software with Finite State Machines: A Practical Approach. Number 0-8493-8086-3. Taylor & Francis Group, LLC, 1 edition, 2006.
13