Viral polymorphism

Stephen Pearce
SANS Institute
October 2003

A paper submitted in partial fulfillment of the requirements for the GSEC Version1.4b.

Abstract

This paper is an overview of polymorphic and metamorphic viruses. It defines them, provides some information regards the safe handling of them and comments on the legality/morality/policy regard the analysis of them. It looks at their history and the methods that they used both with reference to individual viruses and the virus toolkits prevalent in the early 90s. The response of the anti-virus industry is described along with the more recent evolution to metamorphic viruses and the challenge they provide. The aim will be to describe the techniques and then draw parallels between what was seen with viruses and what may happen with worms which now dominate the "virus" world.

Introduction

This paper will consider the general topic of viral hiding methods. In particular it will consider the methodology adopted for polymorphism and the toolkits that support it. In looking at viruses, it will be natural to consider ethics, safety and other factors in such work. A potted history of viruses and viral polymorphism will be given as it is the author's contention that the lessons of history for viruses may be repeated with worms.

Ethics

The quote [1] below from the documentation within a virus writing guide indicates some of the thinking that exists within the virus community. It underlines the ethical issues that exist regard the topic of viruses.

> Virii are wondrous creations written for the sole purpose of spreading and destroying the systems of unsuspecting fools. This eliminates the systems of simpletons who can't tell that there is a problem when a 100 byte file suddenly blossoms into a 1,000 byte file. Duh. These low-lifes do not deserve to exist, so it is our sacred duty to wipe their hard drives off the face of the Earth. It is a simple matter of speeding along survival of the fittest.

With the analysis of viruses and their properties, you must consider your actions carefully. First and foremost there is the legal position. In general it is the act of sending the virus into world at large that is illegal. The punishments for this can be quite severe. Most countries, such as the USA, allow virus creation under the right to Freedom of Speech. However these observations are a personal view and IN NO WAY a legal opinion.

Within the context of membership of SANS, one must also consider the ethics of the situation. This is spelled out in the GIAC Code of Ethics [2] which includes the following statement,

> I will not engage in or be a party to unethical or unlawful acts that negatively affect the community, my professional reputation, or the information security discipline.

I would argue that the key factor is whether the work done is focused on improving the

knowledge regards viruses for the defense or gives greater help to those who would want to write viruses. In writing this paper, my intention is to educate and expose the methods used within viruses for defense and not to aid those who might write viruses. Thus I will not include any source code within this paper. In truth there are many guides on how to write viruses. It is highly unlikely that any prospective ill-intentioned virus writer would look to the SANS reading room as a resource.

An important issue when working with viruses is to ensure that you do not become a victim yourself. They are designed to replicate and thus careful protective measures need to be taken. The credo of defense in depth is an obvious answer here. Precautions should be taken while analyzing viruses. The first is to backup any data you would wish to preserve. It is sensible to work on an isolated system which at the end of the research can be scrubbed clean i.e. the operating system re-installed.

An alternative to that is to retain the concept of an isolated system but do it virtually. When investigating the replication properties of viruses, one of the virtual operating system products such as VirtualPC or VMWare would be suitable. Working within a virtual system, the virus is constrained to that and can be deleted by deleting the virtual operating system. Alternatively the operating system could be marked to reject changes. I would err on the side of caution always when dealing with viruses. A tool to perform secure deletion was my choice.

When conducting work on or searches about a subject such a virus technology, it is worth highlighting that the importation of viral tools and live viruses is very likely prohibited within your work environment. Some of the information resources that can be found are on web sites that also host other material of a less than salubrious nature. Thus it is essential to have clear permission prior to any research regards viruses.

Definition

It is useful always to define terms for the subject of discussion. We will take Dr. Fred Cohen's definition [3] for viruses:

> We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected may also act as a virus and thus the infection grows.

Within the framework of this definition, I will take the view that worms are a form of virus. This can be (and has been) argued both ways. It can be rationalized by considering a worm as a virus infecting the operating system. I take this pragmatic view that a worm is a virus since worms have become the most common viral incident.

This can be seen in the Virus Prevalence data from the latest issue of Virus Bulletin [4]. For July 2003, we see that worms are the most prevalent form of viral incident reported. Nearly two thirds of incidents were from two worms W32.Sobig and W32.Opaserv. Looking at the top ten viruses, which account for over 90% of all incidents, six were worms and they provided 81% of all incidents. A similar story can be found in the web pages for the Wildlist [5], another resource for the measurement of viral activity.

Thus it is clear that viruses remain a threat to systems. In general, the virus wishes to hide its code both on disc and in memory, to conceal its actions and to fake dates/times/checksums. The longer it can remain hidden for, the more likely it will be able to replicate.

Many techniques have been used to achieve these aims. The one I will focus on is that of polymorphism. The intention here is to hide the code so that it is not identified as a virus.

The diagram below shows a simplified viral infection. The viral code has placed itself at the end of the original code. It has also altered the entry point so that it points to the viral code. The viral code may have stored the original entry point for the original code so that it can then

return control to that code. Within the viral code there may be two trigger points; one to decide whether to attempt to infect another executable and the other to decide what payload, if any, should run.

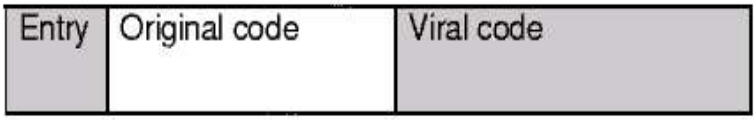

Figure 1: Basic Virus

The first improvement to this design was to try and remove most of the viral code from view. The code was encrypted and a small decryption routine placed before it. The entry point points at the decryption code. The main viral code is now hidden and is only revealed when the decrypt routine runs on it. Polymorphism is defined as state of having many or various forms. It applies here in that a polymorphic virus will have many different decrypt codes available. The simplest version (and referred to as Oligomorphic which means having few or little forms) is to have a finite set of constant decryptors. The true polymorphic virus has a number of different ways of obscuring the decryption routine.

Within the encrypted viral code, there is code to perform the polymorphic transform. Thus a newly infected file contains its own copy of the viral code complete with the polymorphic engine but under a new decryption. This process continues with the next infection and another new decryption method.

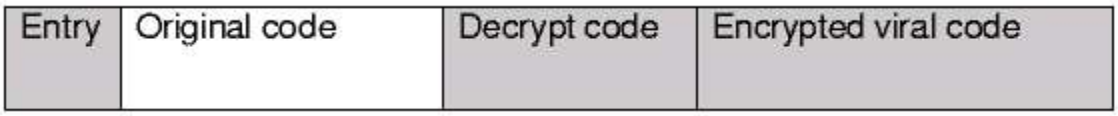

Figure 2: Polymorphic Virus

Figure 3 shows the evolution to a metamorphic virus. The idea here is that there are little islands of viral code scattered in amongst the original code of the tainted executable. When the virus infects a new executable, the islands of code may appear in different places and have different natures modified in similar ways to those possible for the decryption routine in the polymorphic virus.

In this case, it is possible for the entry point to remain bound to the original code. The virus code seizes control within the context of the execution of the original code. This technique is one example of what is referred to as Entry Point Obfuscation and means that the anti-viral technique cannot concentrate on only searching the beginning of the executable to look for signs of the virus.

Figure 3: Metamorphic Virus

History of techniques

Looking back over the history of viruses, and focusing on the hiding techniques used, we see an evolution to the metamorphic virus. The history will focus on the viruses seen in the wild and will cover major events and those that have bearing on the advances in polymorphism. A number of the ideas and techniques were in the gift of anti-virus researchers before the virus writing community put them in the public domain. In 1984, there were viral experiments going on Unix systems while in 1987, a polymorphic virus (V2P2) had been created in the lab.

We begin our story in 1986. The virus had been defined and demonstrated in the academic world but the release of Brain, a boot virus, in Pakistan is agreed as the first PC virus in the wild.

In 1988 the Cascade virus was found in Germany. It used a fixed encryption to hide its contents as was shown in Figure 2. This was also the year that the Morris worm became the first Internet virus. By 1990 virus writers began utilizing many different techniques to try to keep ahead of the anti-virus industry. These included polymorphism in such viruses as 1260. In 1991, the methods were being combined. An example was Tequila which was not only polymorphic but could infect both executables and the Master Boot Record (multipartite) and had some Stealth capability. It would report fake information about its size.

By 1992, there were tools to make virus writing easier. This was the start of the virus writing toolkits. These were either aimed at creating a virus from scratch (Virus Creation Lab) or providing polymorphic functionality that could be linked to an existing virus (Dark Avenger Mutation Engine).

In 1996 the emphasis changed as macro viruses came to the fore with firstly Concept which infected Word documents and later evolved to Excel with Laroux and then into infectors across a range of Office components, Triplicate. Melissa then came along in 1999 and combined macro virus with use of mass mailer replication. Worms have continued to dominate; many of them evolving into blended threats. An example in 2001 was Nimda which used mass-mailing, exploits against IIS web servers and attacks against network shares.

However the techniques utilized in the early 90s to write polymorphic viruses under DOS are seen in 32-bit viruses. First we will give more detail regards the virus toolkits that became prevalent prior to giving a more detailed description of the techniques used in polymorphic and metamorphic viruses.

Toolkits

One of the events in the history of the evolution of polymorphic viruses was the creation of toolkits. These were both for the creation of a virus (e.g. Virus Creation Laboratory) and for the addition of polymorphic functionality to a virus (e.g. Mutation Engine).

The first polymorphic generator was the Mutation Engine. This was at heart a small object file which when linked in with a virus would make a new polymorphic virus. The code allows the user to provide his own random number generator or use one provided with the Engine. Documentation provided with the Engine described how it could be used and included a demonstrator virus using the Engine.

There followed a number of other polymorphic generator tools such as TridenT Polymorphic Engine, NuKE Encryption Device, Dark Angel's Multiple Encryptor. Both NED and DAME were distributed as source code. Unsurprisingly a number of polymorphic viruses were found in the wild based upon these engines. The very fact that these engines needed to be distributed to be used meant that the anti-virus companies were ready for such generated viruses.

Methods

Many of the ideas here began in the DOS world and have been taken across to the 32-bit world. The idea in polymorphism is to change the look of the code without changing its functionality. In the 80x86 architecture, some instructions can make use of a range of registers. For example to clear a register to zero, you can XOR together either AX or BX. Similar tricks can be done to index an array with a SI or DI. The virus, Regswap, did this.

A further method would be to replace an existing instruction or add null instructions. They could be instructions that do nothing such assigning the contents of a register to itself. Alternatively you can replace an existing instruction with a functionally equivalent one. For example, you can increment (INC) an index, you can add (ADD) one or become more convoluted with add two then subtract one (ADD then SUB). Other methods include using

intermediate registers to move values through prior to assigning them to their final destination.

We can also make changes to the order of instructions with the use of jump instructions. This is shown below:

```
        Instruction_A
        Instruction_B
        Instruction_C
```

This could be transformed into:

```
        Jump Label_1
Label_2:
        Instruction_C
        Jump Label_3
Label_1:
        Instruction_A
        Instruction_B
        Jump Label_2
Label_3:
```

These engines were transforming low-level code and were seen in 2000 in the viruses, Zperm and Evol. It is possible, assuming a compiler is available, to do the same with high-level code. Here the same sorts of transformations are applied and the code re-compiled resulting in a transformed executable. This was done in 2000 by the virus, *Apparition*.

These transformations originally were intended to change the nature of the decryptor code for the polymorphic virus. However the evolution to a metamorphic virus was predictable. There was a small diversion on the way with Entry Point Obscuring viruses, an example being Rainsong. Here the virus does not use the entry point at the start of the infected file but inserts its start point at a likely point in the executable. This virus (as do the later metamorphic viruses) relies on gaining control during the execution of infected executable. The EPO methods are varied and include compressing the existing code, providing a decompression code and then using the newly created space for its own code. Alternatively it is possible to add new sections to the executable or identify & use space within an existing executable.

In 2001 the technique (described in figure 3) to spread through an executable was seen in the virus, *Zmist*. These techniques were combined in 2002 in the virus, *Simile*, which has a metamorphic engine using many of the techniques above. The engine is hugely complex and account for most of the 14,000 lines of assembly code [6].

Anti-viral methods

There are a number of ways by which anti-viral techniques can find a virus. The virus scanner is the method most commonly identified with virus detection. This looks at data in memory or on disc and decides whether it is a virus. As in Intrusion Detection, they are greatly concerned with the problem of False Positives and False Negatives. A False Negative will have meant that the scanner has not alerted when a virus was present. By its nature, the virus then may spread and the problem can become somewhat of a vicious cycle. On the other hand, a False Positive will mean that there has been a virus alert but there was no virus. The possibility of this resulting in self-inflicted damage is high.

The scanner has distinct patterns that it searches for which typify particular viruses. This pattern matching has had to evolve to counter some of the techniques the virus writers have used. The scanner also has other constraints such as its speed. The scanner can only detect things it has been given the pattern for. Thus it is limited to existing viruses that the anti-virus company has created a distinct pattern for. It is also required that the customer has kept up to date with these patterns. Scanning is only part of the anti-viral toolkit.

Integrity checkers take a different approach and are a protective technique. They function like the Tripwire tool. They checksum files and will alert when they change. This means they will catch known and unknown viruses by their actions in infecting files. However they are also

prone to problems of False Positives. As an example of a False Positive, it may be that a legitimate upgrade or patch was applied to the file in question rather than any viral activity. This is compounded by the possibility that a Slow (where a virus waits for legitimate write to occur and piggybacks on the event) virus may use the event of patching, as cover for its own changes. In this case, a user may assume she faces a False Positive when in truth a virus was at work.

The final category of anti-viral tools has the title of heuristics. This relies on the observation of generic activity that characterizes a virus. An example would be when scanning an executable, if the code were self-modifying, searches for executable files, takes no parameters then each of these properties may contribute some score towards a positive identification as a virus. These scores could each be different and the overall total measured against some threshold that characterizes a virus. The choice of the various activities and the setting of their associated scores is at heart an attempt to characterize the general qualities that a virus may have.

Unlike the standard scanner, the heuristic scanner does not declare a particular virus is present; the heuristic scanner can only say that it believes some kind of virus is present. However it does potentially find new viruses. It also suffers from the problems of False Positives and False Negatives. Some of the things that characterize a virus may be also be true of a self-extracting compressed file. Likewise a true virus may include non-viral activity to try to fool the heuristic scanner.

Within most anti-viral packages, there is also provision to prevent viral activity in a heuristic manner. This activity checker will alert if it sees the actions such as a write to the Master Boot Record. It has not found a virus just activity that is indicative of one.

Evolution of anti-viral methods

The methods above, particularly the scanner, have had to evolve to deal with the advances in the polymorphic and metamorphic engines. The pattern matching algorithms had to become more sophisticated. The switching of registers required scanners to have wildcard for particular positions. The use of redundant instructions meant the scanner needed variable length wildcards. Eventually the patterns within the decryption code became so short and spread out that a different approach was needed in the anti-viral toolkit.

One method against the more advanced polymorphic engines was to run the code within an emulator to allow the decryption to uncover the true viral code. This provides for the precise identification of viral type (necessary for the safe disinfection of the code). The virus writers did produce a number of anti-emulation tricks including rarely-used instructions that they hoped the emulator would not provide. The problem with using an emulator is that it is time-consuming and thus needs to be focused on the executables that most likely contain viruses. This provides a natural synergy with the heuristic scanners.

Future

Given the history of the inventiveness of the virus writers and the apparent problems of metamorphic viruses, it is clear that the anti-virus techniques will continue to need to move forward. Though the theory provides for undetectable viruses, pragmatism has allowed us to detect viruses for some time now. Given the techniques in use are generally code obfuscation then it would seem natural to look towards code simplification techniques to unravel the viral routines and reveal their true nature.

It does not seem a tremendous leap to put the present worms at the level of early viruses. Looking ahead to worm construction kits and metamorphic generators for worms is not a pleasant matter.

References

1. Dark Angel's Phunky Virus Writing Guide, URL: http://vx.netlux.org/lib/static/vdat/tuda0001.htm
2. GIAC Code of Ethics, URL: http://www.giac.org/COE.php
3. Frederick B. Cohen; Computer Viruses, Theory and Experiments; 7th Security Conference, DOD/NBS Sept 1984.
4. Virus Bulletin Prevalence table, URL: http://www.virusbtn.com/resources/prevalence/index.xml
5. The Wildlist, URL: http://www.wildlist.org/WildList/200307.htm
6. Ferrie, Peter and Szor, Peter. "Zmist Opportunities." URL: http://pferrie.tripod.com/vb/zmist.pdf

Booklist:

- Ferbrache, David. "A Pathology of Computer Viruses." Springer-Verlag, 1992
- Grimes, Roger A. "Malicious Mobile Code." O'Reilly, 2001

Virus descriptions sourced from:

- Virus Encyclopedia. URL: http://www.viruslist.com/eng/viruslist.html
- Virus Bulletin Resources. URL: http://www.virusbtn.com/resources/viruses/index.xml
- Virus Information library. URL: http://vil.nai.com/vil/default.asp
- Ferrie, Peter. "Peter Ferrie homepage papers and publications." URL: http://pferrie.tripod.com/
- Szor, Peter. "Peter Szor homepage papers and publications." URL: http://www.peterszor.com/

Virus history was sourced from:

- Kaspersky, Eugene. "Virus History by Eugene Kaspersky." URL: http://www.viruslist.com/eng/viruslistbooks.html
- Slade, Robert M. "History of Computer Viruses." URL: http://www.cknow.com/vtutor/vtsladecontents.htm
- Solomon, Dr. Alan. "A Brief History of PC Viruses." URL: http://www.cknow.com/vtutor/vtsolomancontents.htm

Particular study was taken from:

- "Polymorphism resources." URL: http://vx.netlux.org/lib/static/vdat/miscella.htm#polyinvir
- "Viral mutation engines." URL: http://www.hackpalace.com/virii/engines/
- Dark Angel. "Dark Angel Mutation Engine Object Files and Documentation."
- Bontchev, Vesselin. "Future Trends in virus writing." URL: http://www.virusbtn.com/old/OtherPapers/Trends/
- Szor, Peter and Ferrie, Peter. "Hunting for Metamorphic." Virus Bulletin Conference, September 2001.
- Szor, Peter. "Attacks on Win32 part 2." Virus Bulletin Conference, September 2000.