The evolution of self-defense technologies in malware

Alisa Shevchenko, Senior Analyst Kaspersky Lab.

Introduction

This article explores how malware has developed self-defense techniques and how these techniques have evolved as it has become more difficult for viruses to survive. It also provides an overview of the current situation.

First we must define the meaning of the term “malware self-defense", which is not as unequivocal as it may seem at first glance. When malware attacks antivirus programs, this is clearly a form of self defense. When malware covering its tracks, this is also in some sense a form of self defense, although less obviously so. An even less obvious form of self defense is the very evolution of malicious programs. After all, one of the motivations behind virus writers searching for new platforms that can be infected and for new system loopholes is to spread new viruses in the wild, into areas where no one yet bothers to look for malicious code as nothing has been found there before.

In order to avoid confusion about what is considered a self-defense technology and what is not, this article examines only the most popular and obvious means of malware self-defense. First and foremost this includes various means of modifying and packing code, in order to conceal the presence of malicious code in the system and to disrupt the functionality of antivirus solutions.

Classifying malware self defense

There are many different kinds of malware self-defense techniques and these can be classified in a variety of ways. Some of these technologies are meant to bypass antivirus signature databases, while others are meant to hinder analysis of the malicious code. One malicious program may attempt to conceal itself in the system, while another will not waste valuable processor resources on this, choosing instead to search for and counter specific types of antivirus protection. These different tactics can be classified in different ways and put into various categories.

As the goal of this article is not to create a strict classification system for malware self-defense techniques, let's consider a classification system that will provide an understanding of this issue at an intuitive level. We take the two criteria which we believe are the most important, and from there we will create a scatter plot with two axes representing those two criteria.

The first criterion is a malicious program's level of self-defense activity. The most passive malware does not attempt to defend itself in any way, i.e. it does not contain any such code. Instead, the author creates a kind of protective shell for the program. More active self-defense systems involve deliberately aggressive techniques.

The second criterion is the degree to which a malicious program's self-defense mechanism is dedicated. The most narrowly dedicated forms of self defense are found in malicious programs that somehow disrupt the function of a specific antivirus program. More general self-defense mechanisms are designed to defend malicious programs against everything by making the virus presence in the system as undetectable as possible in every way.

We have used a scatter plot to present the different kinds of malware self-defense mechanisms. This diagram is merely a simple example that we can use as a guide to categorize different means of malware self-defense. This model is based on a careful analysis of malware behaviors,

but is, necessarily, subjective.

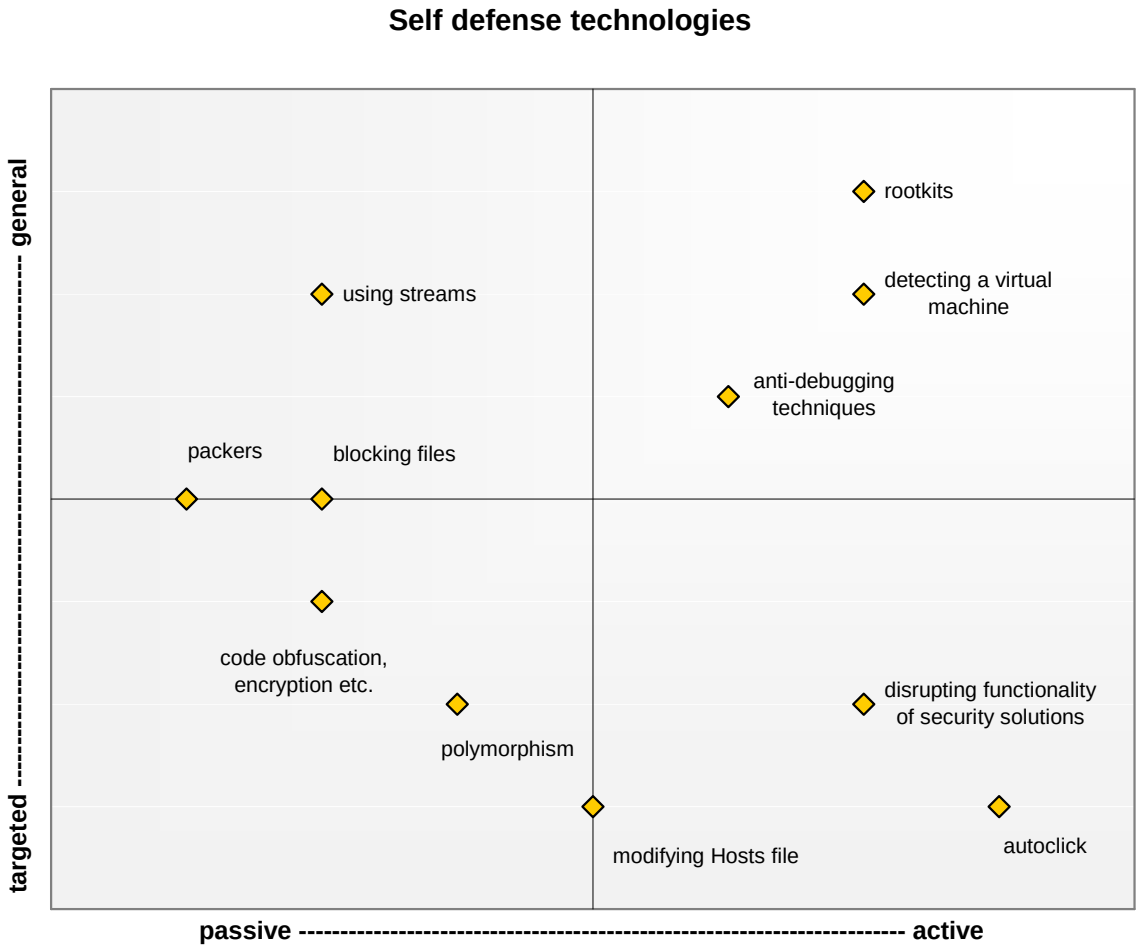

Figure 1. A scatter plot of malware self-defense technologies

Malware self-defense mechanisms can fulfill one or more tasks. These include:

1. hindering detection of a virus using signature-based methods;
2. hindering analysis of the code by virus analysts;
3. hindering detection of a malicious program in the system;
4. hindering the functionality of security software such as antivirus programs and firewalls.

This article will only examine malicious programs written for the Windows operating system (and its predecessor, DOS) due to the rarity and relatively small number of malicious programs for other platforms. All of the trends examined in this article that apply to executable malware files (EXE, DLL and SYS), and to some extent also apply to macro viruses and script viruses, which is why I will not be addressing the latter separately.

Sources: polymorphism, obfuscation and encryption

It makes sense to examine polymorphism[1], obfuscation[2] and encryption together, as they all fulfill the same end, albeit to different degrees. Initially, modification of malicious code had two goals: to make it more difficult to detect files and to make it more difficult for virus analysts to examine the code.

The history of malware began in the 1970s, but the history of malware self-defense didn't start until the late 1980s. The first virus that attempted to defend itself from the antivirus utilities then in existence was the DOS virus Cascade (Virus.DOS.Cascade). It defended itself by partially encrypting its own code. This wasn't very successful, however, since each new copy of the virus - despite being unique from previous copies - still contained an unaltered piece of code that gave it away every time. As a result, antivirus programs could still detect it. Nevertheless, virus writers were turning in a new direction, and in two years the first polymorphic virus appeared: Chameleon (Virus.DOS.Chameleon). Chameleon, also known as 1260, and its contemporary Whale, used complex encryption and obfuscation methods to protect their code. Two years later, we saw the emergence of so-called polymorphic generators, which could be used as out of the box defense solutions for malicious programs.

Why code modification can be used to hinder file detection, and how file detection works, needs some explanation.

Until recently, antivirus programs worked exclusively by analysis file code. The earliest signature-based detection methods focused on searching for exact byte sequences, often at a fixed offset from the beginning of the file, in a malicious program's binary code. Later heuristic detection methods also used file code, but with a more flexible, probability based approach to searching for common malware byte sequences. Obviously, it's not difficult for malicious programs to get around that kind of protection if each copy of the program includes a new byte sequence.

This task is fulfilled by the application of **polymorphic** and **metamorphic** techniques, which essentially - without getting into the technological nitty-gritty – enable a malicious program to mutate at byte level when the program creates a copy of itself. Meanwhile, the program's functionality remains unchanged. **Encryption** and **obfuscation** are primarily used to hinder code analysis, but when they are implemented in a certain way, the result can be a variation of polymorphism – an example here is again Cascade, where every copy of the virus was encrypted with a unique key. Obfuscation may just hinder analysis, but when it is applied in a different way to every copy of a malicious program, it hinders the effective use of signature-based detection methods. However, it cannot be said that any one of the abovementioned tactics is more effective than any other in terms of malware self-defense. It would be more correct to say that the effectiveness of these techniques depends on the specific circumstances and how the techniques are implemented.

The use of polymorphism only became relatively widespread in terms of DOS file viruses. There's a reason for this. Writing polymorphic code is a highly time-consuming task that is really only justified in cases when a malicious program is self-replicating: then each new copy contains a more or less unique byte sequence. The majority of contemporary Trojans aren't able to self-

---

[1] Polymorphism - a technology that allows a self-replicating program to fully or partially modify its outward appearance and/or the structure of its code during the replication process.

[2] Obfuscation - a combination of approaches used to obscure the source code of a program. This is designed to make the code as difficult as possible to read and analyze it while retaining full functionality. Obfuscation technologies can be applied at the level of any programming language (including high level, script and assembler languages). Examples of very simple obfuscation include adding neutral instructions (which do not alter program functionality) to the code or making the code harder to read by using an excessive number of unconditional skips (or unconditional changeovers disguised as conditional skips).

replication, and polymorphism is therefore irrelevant. That's why since the end of the DOS file virus era, polymorphism has been seen less, and it was used mostly by virus writers who wanted to show off their skills rather than to create a particularly useful malicious function.

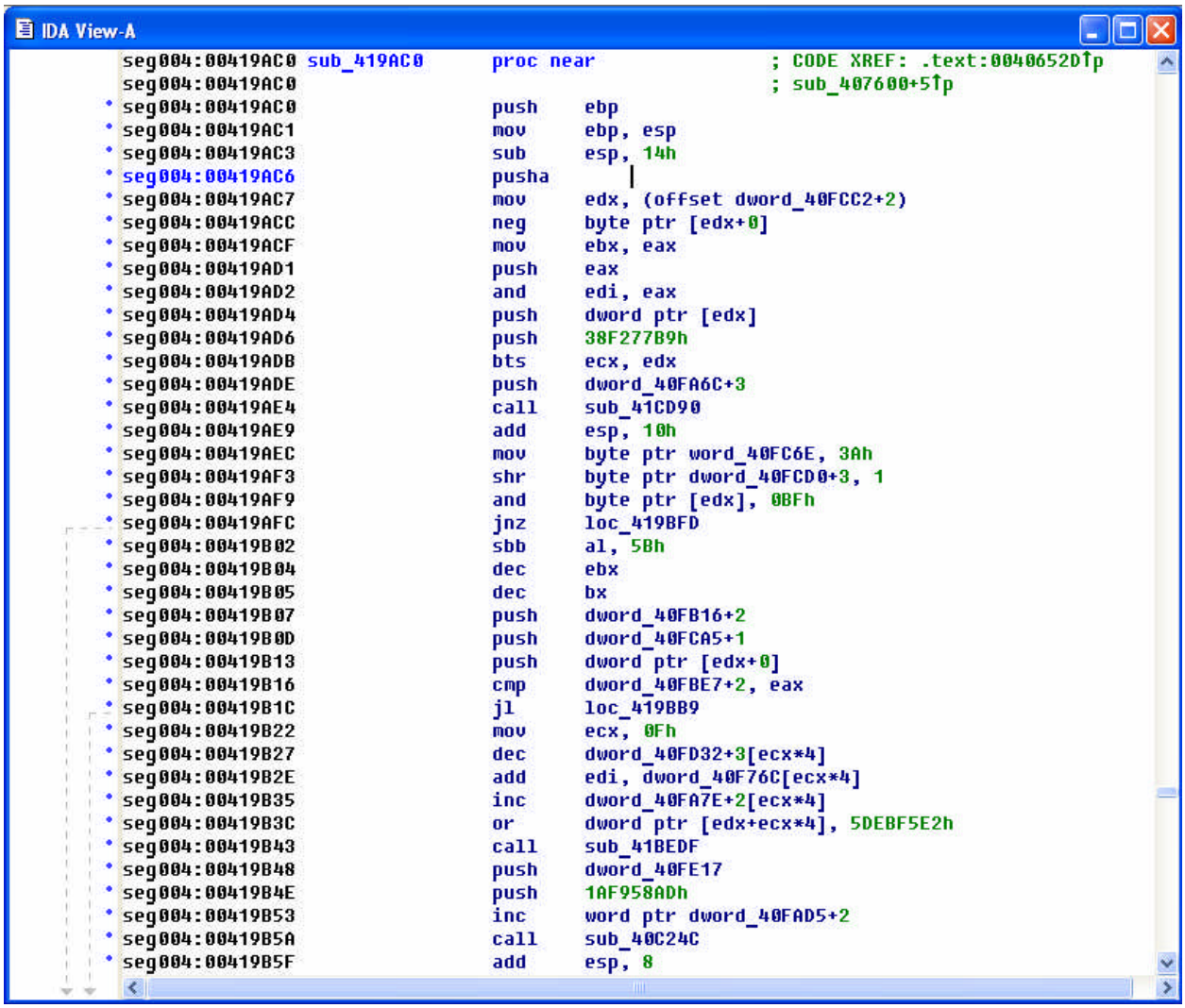

Figure 2. The polymorphic code of P2P-Worm.Win32.Polip

In contrast, obfuscation continues to be used today, as are other code modification methods that, to a large extent, make it more difficult to analyze code as opposed to hindering detection.

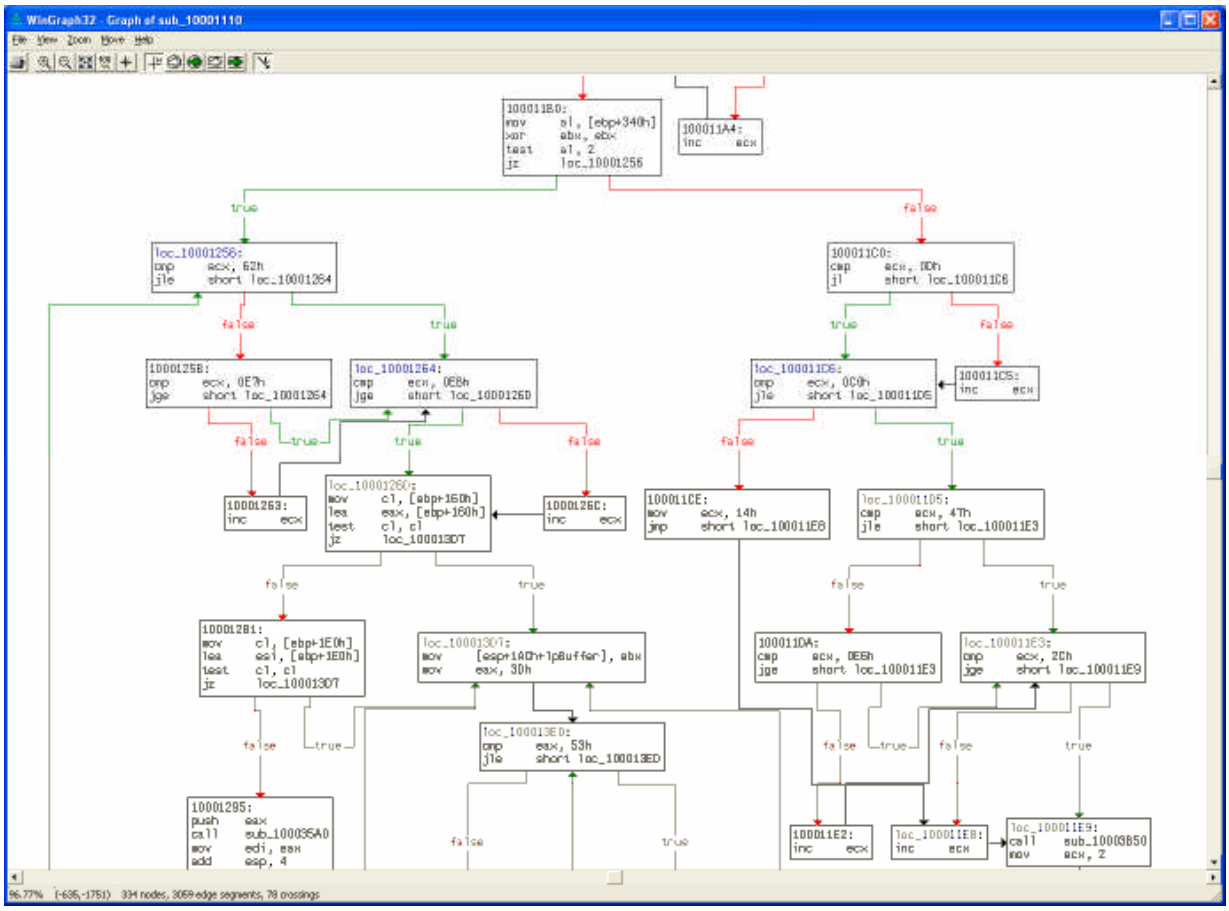

Figure 3. A diagram of the obfuscated code in Trojan-Dropper.Win32.Small.ue

Since behavioral detection methods arrived and began to squeeze out signature-based methods, code modification techniques have become less useful in hindering malware detection. This is why polymorphism and related technologies are not commonly used today and are only really a means for hindering the actual analysis of malicious code.

Stealth viruses

Concealing malicious programs in the system became the second method of self-defense against detection that was mastered by virus writers in the DOS era. This technique was first used in 1990; to be more precise, it was included in the arsenal of a virus we have already mentioned - Whale. Essentially, the concealed virus would one way or another intercept DOS system services and pass false data to the user or the antivirus program – for instance, "clean" boot sector contents, instead of the real contents which had been infected by the malicious program.

Stealth technologies for the DOS operating system were reborn as rootkit technologies for the Windows operating system 10 years later.

The mechanisms used to conceal viruses in the system are covered in more detail in the Rootkit section.

Packers

Gradually, viruses - malicious programs that can function only within a victim body and which are unable to exist as a separate file - are being replaced by Trojans, which are fully independent malicious programs. This process began when the Internet was still slow and more limited than it is today. Hard disks and floppy disks were small, which meant that the size of a program was very important. In order to reduce the size of a Trojan, virus writers began to utilize so-called

packers - even back in the DOS era. Packers are dedicated programs that compress and archive files

A side effect of using packers that can actually be useful from a malware point of view is that packed malicious programs are more difficult to detect using file methods.

When creating a new modification of an existing malicious program, the virus writer usually changes several lines of code, while leaving the heart of the program untouched. In the compiled file, the bytes for a certain sequence of code will also be altered and if the antivirus signature does not include that very sequence, then the malicious program will still be detected as before. Compressing a program with a packer solves this problem as changing even just one byte in the source executable results in an entirely new byte sequence in the packed file.

Figure 4. The visible difference between packed and unpacked code

Packers are still commonly used today. The variety of packing programs and their level of sophistication continue to grow. Many modern packers, in addition to compressing a source file, also equip it with additional self-defense functions aimed at hindering the unpacking and analysis of the file using a debugger.

Rootkits

Malicious programs for the Windows operating system started using stealthing technologies to hide their presence in the system in the first years of the new millennium. As mentioned above, this was approximately 10 years after stealthing programs appeared as a concept and was implemented for DOS. In early 2004, Kaspersky Lab encountered a surprising program that couldn't be seen in the Windows processes and files list. For many antivirus experts, this was a new beginning – understanding stealthing technologies for malicious programs for Windows – and it was the harbinger of a major new trend in the virus writing industry.

The term “rootkit” stems from Unix utilities that are designed to provide a user with unsanctioned root access within the system without being noticed by the system administrator. Today, the word rootkit covers dedicated utilities used to conceal information in the system, as well as malicious programs with functionality which enables them to mask their presence. These

include the manifestations of any third-party registered applications: a string in the list of processes, a file on disk, a registry key or even network traffic.

How do rootkit technologies which are designed to conceal malicious programs in the system make it so difficult to detect the malicious programs using antivirus or other security software? It's very simple: an antivirus utility is an external agent just like the user. Generally, if a user can't see something, then an antivirus program can't see it either. However, some antivirus solutions implement technologies which sharpen their vision, enabling them to detect rootkits when users cannot see them.

A rootkit is based on the same principle as DOS stealth viruses. A large number of rootkits have mechanisms which modify a chain of system calls (Execution Path Modification). This kind of rootkit may act as a hook located at a certain point of a route along which commands or information are exchanged. It will modify these commands or information in order to distort them or control what happens on the recipient's end without the recipient's knowledge. Theoretically, the number of points at which a hook can be located is limitless. In practice, there are currently several different methods commonly used to hook APIs and kernel system functions. Examples of this kind of rootkit include the widely known utilities Vanquish and Hacker Defender and malicious programs such as Backdoor.Win32.Haxdoor, Email-Worm.Win32.Mailbot, and certain versions of Email-Worm.Win32.Bagle.

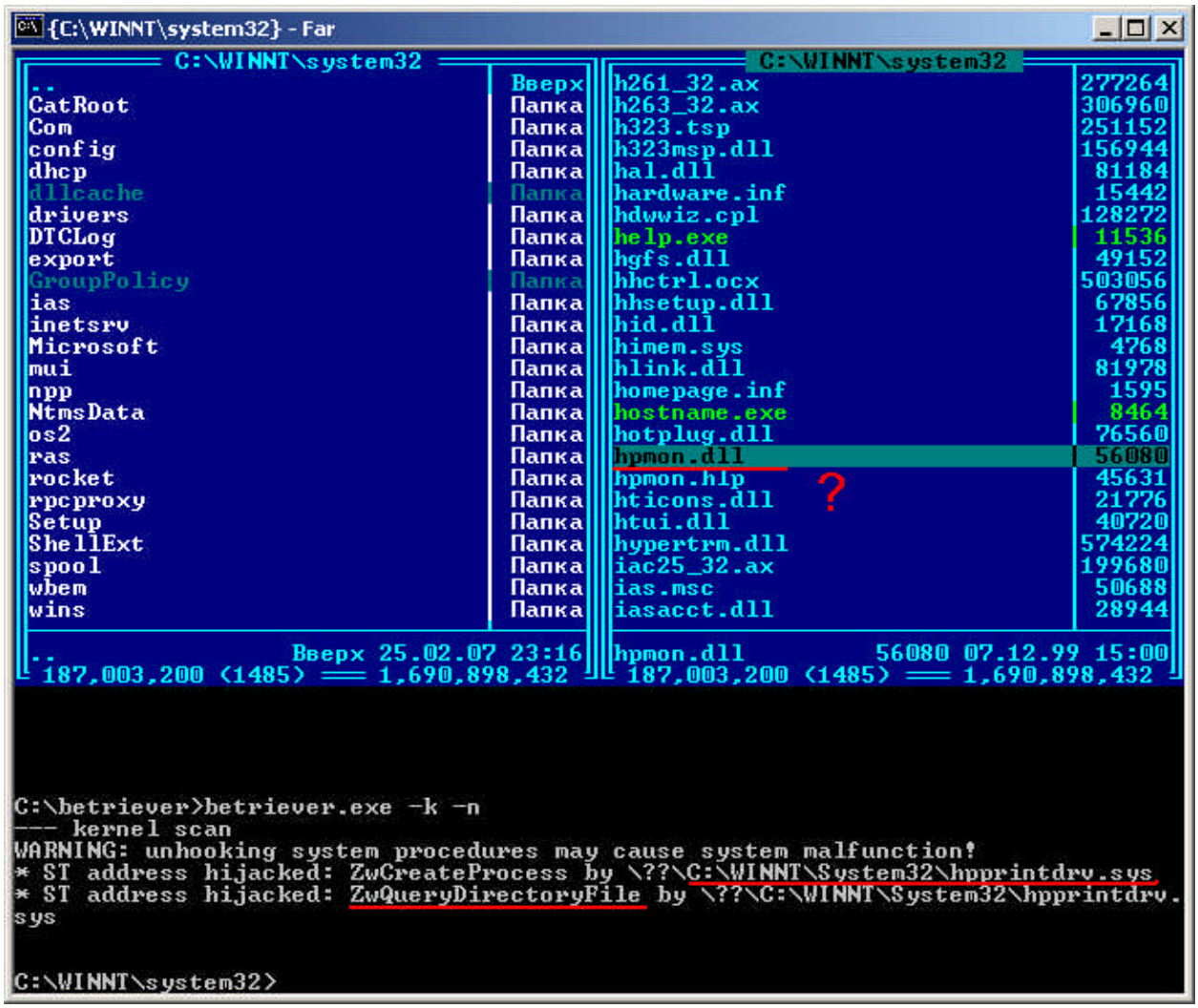

Figure 5. Hooking ZwQueryDirectoryFile conceals the driver file in the list of files.

Another common type of rootkit technology is Direct Kernel Object Modification (DKOM), which can be viewed as an insider that modifies information or commands directly in their sources. These rootkits alter system data. A typical example is the FU utility; the same functions can be found in Gromozon (Trojan.Win32.Gromp).

A newer technology that officially corresponds to the rootkit classification conceals files in alternate data streams (ADS) in NTFS file systems. This technology was first implemented in 2000 in the malicious program Stream (Virus.Win32.Stream), and got a second wind in 2006 in the form of Mailbot and Gromozon. Strictly speaking, exploiting ADS is not so much a means of tricking the system as of taking advantage of little-known functions, which is why this particular technology isn't likely to become very widespread.

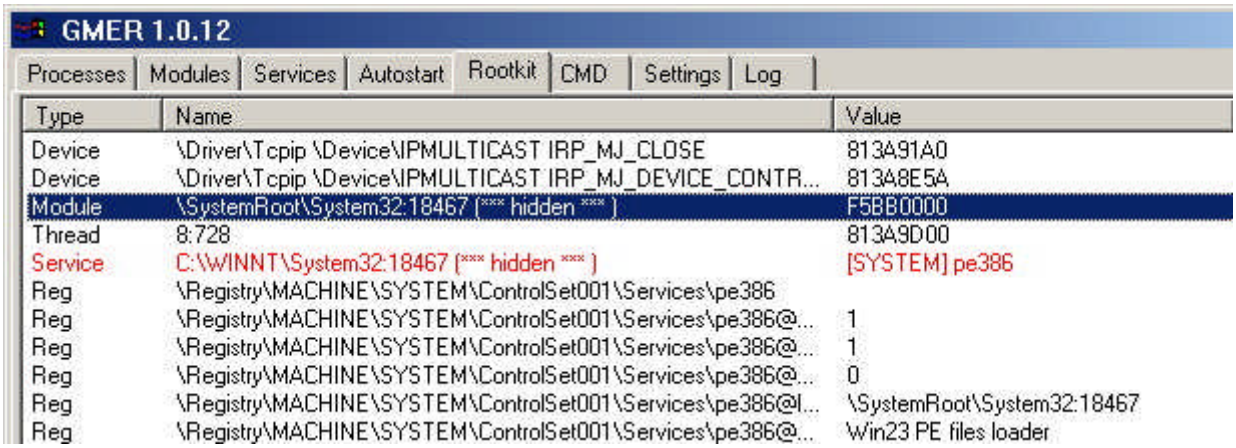

Figure 6. The malicious program Mailbot (Rustock) exploits the system directory stream.

There is another rare technology which only partially falls into the rootkit category (but it corresponds even less to the other classes of malware self-defense examined in this article). This technology uses bodiless files - this means malicious programs do not have any body whatsoever on the disk. There are currently two known representatives of this subgroup: Codered, which emerged in 2001 (Net-Worm.Win32.CodeRed) which exists in this form only within the context of MS IIS, and a recent proof of concept Trojan that stores its body in the registry.

The modern rootkit trend aims towards the virtualization and use of system functions – in other words, penetrating even more deeply into the system.

Combating antivirus solutions

There have always been malicious programs that have actively defended themselves. Self-defense mechanisms include:

- Performing a targeted search of the system for an antivirus product, firewall or other security utility, followed by disrupting the functioning of that utility. An example might be a malicious program that searches for a specific antivirus product in the process list and subsequently attempts to disrupt the functioning of that antivirus.
- Blocking files and opening them with exclusive access as a counter measure against file scanning by the antivirus.
- Modifying the hosts file in order to block access to antivirus update sites.
- Detecting query messages sent by the security system (for example, a firewall window with an inquiry such as "Allow this connection?") and imitating a click on the "Allow" button.

Actually, a targeted attack against a security solution is more similar to the reaction of someone pushed up against a wall rather than an active attack. In today’s conditions, when antivirus

companies analyze more than the code contained in malicious programs - they analyze their behavior as well –malicious programs are more or less powerless. Neither polymorphism, nor packers, nor even stealth technologies will provide malicious programs with total protection. This is why malware has set its sights on certain manifestations or functions of the so-called enemy. Of course, sometimes self-defense mechanisms are the only solution; otherwise they would not be so common, as they pose too many disadvantages from the viewpoint of maximum, full-range defense.

What will the future bring?

Antivirus protection is continually evolving, moving from file analysis to program behavior analysis. In contrast to file analysis, the basics of which were explained in the section on polymorphism and obfuscation, behavior analysis is based not on working with files, but with events at system level, such as "list all active system processes," "create a file with this name in the directory shown," and "open the port indicated to incoming data." By analyzing the chain of these events, an antivirus program can calculate the extent to which the component generating these processes is potentially malicious, and give a warning when necessary.

However, behavior analysis can get confusing when it comes to terminology, and it's not always easy to get things straightened out. For example, a behavioral analyzer may go by different names: HIPS, proactive protection, heuristic, or sandbox… However, regardless of the term, one thing remains clear: malicious programs are ultimately powerless in the face of behavioral analysis. This vulnerability will probably have an influence on the future evolution of malicious programs.

In other words, virus writers are faced with the need to somehow work around behavior analyzers. There is no way to know how they will go about tackling this obstacle. But we do know, for example, that the use of obfuscation at the behavioral level is basically ineffective. The evolution of environmental diagnostics, however, is very interesting. This is because it assumes, in part, a rise in a virus's "self-awareness", which would allow it to determine where exactly it is located: in the "real world" (in a user's clean working environment) or in the "matrix" (under the control of antivirus analysis).

Diagnostic technologies do have their precedents: some malicious programs, if they are launched in a virtual environment (such as VMWare or Virtual PC) destroy themselves immediately. By building this self-destruction mechanism into a malicious program, its author prevents its analysis, which is often conducted within a virtual environment.

Trends and forecasts

Having examined current trends and how effective current approaches are, we can expect the following from the methods of malware self-defense discussed above:

1. Rootkits are moving towards exploiting equipment functions and towards virtualization. This method, however, has not yet reached its peak and probably won’t become a major threat in the years to come, nor will it be widely used.
2. Technology which blocks files on disk: there are two known proof of concept programs that have demonstrated we can expect this area to develop in the near future.
3. The use of obfuscation technologies is insignificant, but nevertheless still current.
4. The use of technologies that detect security utilities and interrupt their performance is very common and widely used.
5. The use of packers is widespread and is growing steadily (both in quantitative and qualitative terms)

6. The use of technologies that detect debuggers, emulators and virtual machines as well as other environmental diagnostic technologies is expected to develop in order to compensate for the mass transition of antivirus products to behavioral analysis.

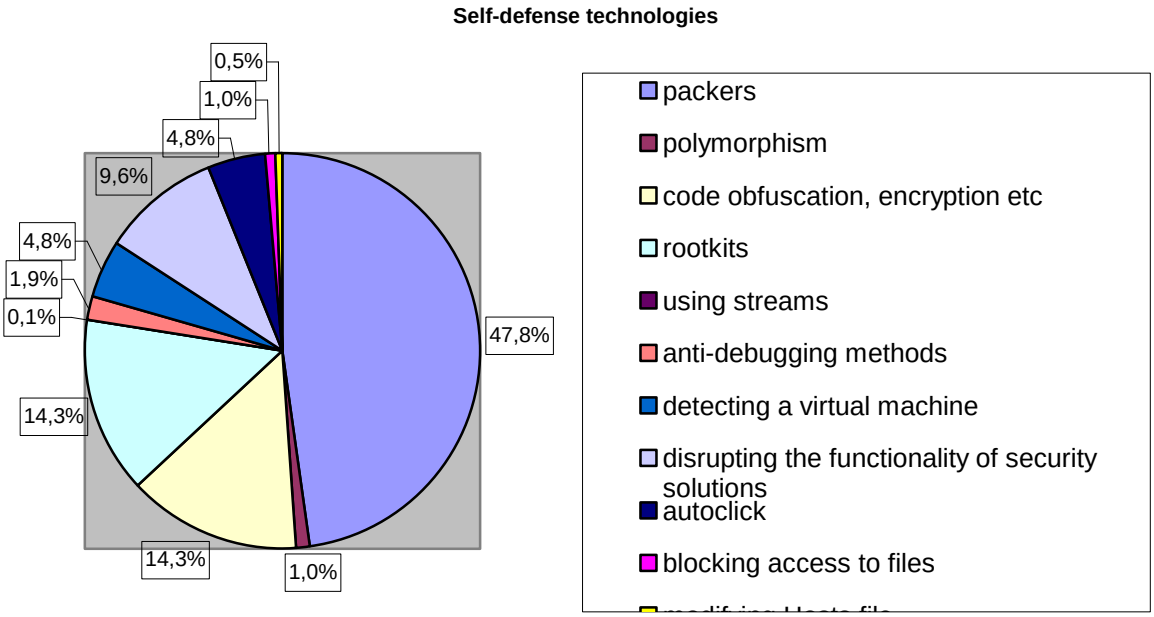

Figure 7. An approximate breakdown of malware self-defense technologies as of early 2007.

It's not difficult to see that the evolution trends in malware self-defense technologies are changing in step with the evolution of malicious programs themselves, as well as protection against malware. When most malicious programs infected files and antivirus programs used signature-based detection, the most prevalent forms of malware self-defense were polymorphism and code protection. Today, malicious programs are mostly independent, and antivirus programs are becoming increasingly proactive. Based on these facts, we can predict which malware self-defense mechanisms will develop more intensively than others:

1. Rootkits. Their invisibility within a system gives them a clear advantage - even if it doesn't prevent their detection. We can most likely expect new kinds of bodiless malicious programs and, a little later, the implementation of virtualization technologies.
2. Obfuscation and encryption. This method will remain common as long is it continues to hinder code analysis.
3. Technologies used to counter security solutions which are based on behavioral analysis. We can expect the appearance of some new technologies, since the ones that are currently being used (targeted attacks against antivirus programs) are not effective. It's possible that we will see some methods of detecting virtual environments or a type of behavioral encryption.

Conclusion

What else can we say in conclusion? The mere existence of malicious programs has led to the existence and development of protection against them. And now we will see the emergence and

development of malware self-defense against protection against malware. This brings up images of a never-ending conflict, where it's not nature, but technology, that is red in tooth and claw.

In the last few years the situation has become increasingly critical. Someone from the cybercriminal underground (or “researchers” dressed in white hats[3]) develops proof of concept code to evade modern means of protection. This person will, for self-promotion purposes and feigning concern about progress, announce that the code is “undetectable”. But we need to emphasize that this concept will be undetectable only at the level of one- or two-step bypass of known security functions, rather than 100% undetectable. It is relatively easy to create a one-step work-around if you are familiar with protection mechanisms.

Such publications cause concern among a certain percentage of users who are not familiar with the way malicious programs and antivirus programs work ("Will my antivirus protect me against this new type of threat?"). In this situation, the people creating self-defense methods have only to chip in a share of their resources in order to restore their reputation and develop a work-around - usually a one-step bypass. In the end, their reputation is saved (of course), the malware - antivirus - user system returns to its initial state, and we are again faced with the same vicious cycle. Each new iteration gives rise to even more sophisticated malware and more heavyweight means of defense.

I want to emphasize that this process uses up a lot of resources, is senseless and endless, and recently, it has become even more exaggerated. All three parties in this conflict need to raise their awareness of the situation. I urge users to expand their understanding, to acknowledge that no means of protection can provide 100% security, and the best way to protect yourself against threats is to prevent them. I challenge those who write viruses to reconsider their intentions and motives behind publishing proof of concept code that will only add fuel to the fire. And I call on the "protectors" to force themselves to think outside the box, and to try to keep more than one step ahead of the enemy. We may not be able to see the end of this so-called arms race, but we can keep it from spiraling out of control.

3

This is a reference to a couple of terms used in computer security jargon: a Black Hat (a malicious hacker) and a White Hat (a security professional who uses hacking techniques for legal, ethical ends).