Parasitic malware: the resurgence of an old threat

Simon Heron, managing director, Network Box UK

In late 2006, an old technique became popular again with the blackhats. They resurrected parasitic malware, a technique whereby malware is added to existing files on a system. This concept has been around since the eighties. One of the earliest viruses, Jerusalem, used parasitic techniques, for example. It infected any .EXE file that it could find, appending its own code to the file so that it could be run and deliver its payload. In 1989, the Datacrime virus infected one .EXE file and one .COM file each time it was run.

The era of macro viruses, email worms and other forms of delivery may have seen this technique die off for a while, but after re-emerging in 2006, this form of malware flourished. McAfee Avert Labs identified 150 new variants of parasitic malware, Philis and Fujacks. New viruses using this technique were also discovered – such as Grum-A – and it seems that this technique has made a comeback. While this form of malware only accounts for around 10% of all malware, it is the sophistication of these viruses that makes them worthy of closer examination.

Simon Heron

It is fully expected that in 2008, blackhats will continue to use this technique to install their malware as one of the attack vectors. The question is why this technique has been resurrected and improved. In order to answer this question, it is important to understand how this technique works and how these viruses propagate. It is also necessary to understand what benefits blackhats derive from this technique and which files are targeted and why. Finally, we will discuss what anti-malware vendors are doing to identify and eliminate this burgeoning threat.

"The term 'parasitic malware' can refer to malware that exploits vulnerabilities in other malware to install itself"

It should also be noted that the term 'parasitic malware' can refer to malware that exploits vulnerabilities in other malware to install itself. For the purposes of this article, however, the term refers to malware that adds itself to legitimate files in one way or another.

It is always useful to consider a number of examples when exploring new malware phenomena, so there follows a brief description of three parasitic malware instances and how they work.

Polip virus

Parasitic malware was making a comeback as early as April 2006, when the Polip virus was discovered. Polip is a polymorphic file infector, which means that it changes its appearance when it infects a file, making standard-pattern matching anti-virus a more difficult task. Polip looks for and infects .exe and .scr files.

It starts by looking in folders given by the operating system variables: %ProgramFiles and %WINDIR%

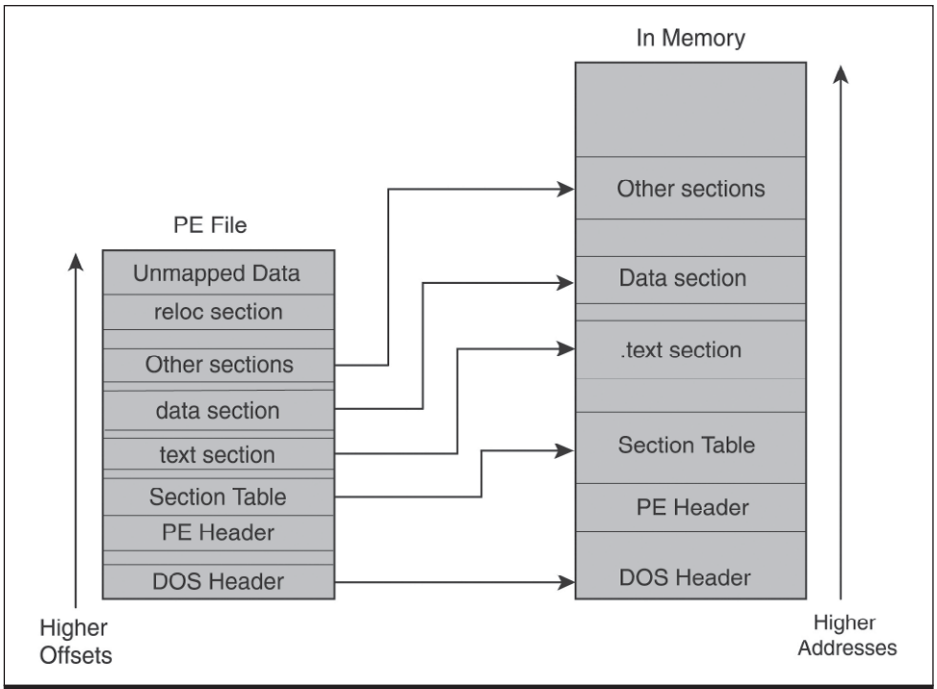

Figure 1: Section in a Microsoft Process Executable File.

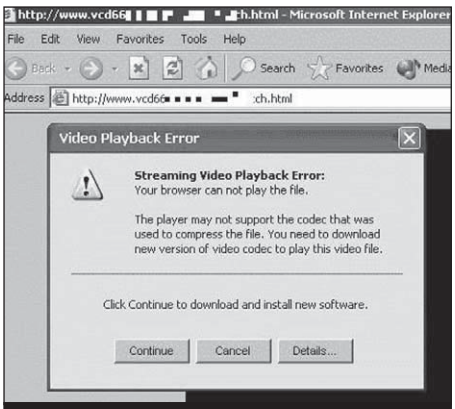

Figure 2: An invitation to infection

folders. It then creates code, or 'hooks', for imported functions for any process it has infected. In this way, any executables used by the infected processes will also be infected. Polip has a number of encryption layers to hide the code; they may not be the most complex, but decryption still takes a long time.

The polymorphic behaviour was sophisticated, ensuring that it changed regularly; and was combined with the ability to create spurious code, which did not have any functionality but did further change the look of the code. It also incorporates anti-debugging and anti-emulation techniques, which made it harder to run the code in emulation and identify how it worked.

The infection was carried out by occupying space in the various sections of the code, contained in the process executable file, which is Microsoft's file format for 32-bit executables and object files. This meant that if it found unused space in the code sections of the file, it inserted code into them where possible without increasing the sizes of those sections.

It also could increase the VirtualSize of the data section of the targeted code (VirtualSize being the figure representing the total size of a section of the PE file). This space was used by the virus for the 'junk' code that it generated to obfuscate and change its appearance. If a resource section was found in the target code, it could shift that section and insert a new one after the last data section and before the resources (the resources section being part of the 'other sections' in figure 1). This would ensure the correct functionality and effectively give more space for its own code without changing the overall size of the target code.

For the processes it infected, it hooked the following APIs by patching directly the kernel copy from each process address space:

CreateFileW
CreateFileA
SearchPathW
SearchPathA
CreateProcessW
CreateProcessA
LoadLibraryExW
LoadLibraryExA
ExitProcess

These hooks allowed the virus to infect all files that an infected process accesses through these APIs.

The virus was able to connect to Gnutella P2P network, acting as a client. It used a predefined list of Gnutella webcache servers, in order to obtain lists of other P2P clients connected at the time. Using the P2P network, it was able to spread itself like a worm. Once a machine on a Gnutella protocol-based client was infected, the virus became available to other machines on the P2P network. If a user copied an infected file and subsequently ran it, their machine then became a host for this virus. Also, if a user received the virus by email and did not have a Gnutella protocol-based client but were foolish enough to run the virus, the virus implemented its own P2P Gnutella client on that machine, causing it to become a host for spreading the virus.

However, despite all these clever features, it still gives itself away. In spite of efforts to hide the malware, infected files still grow in size. It also creates unexpected modification of files on the hard disk, and spreads itself using unexpected connection or activity on the Gnutella P2P network.

Kibik virus

This virus was spread mainly by enticing unwary users to sites hosting an XML exploit: Exploit-XMLCoreSrvcs. This exploit is known as a 'drive-by install', which means a vulnerability in Internet Explorer would allow code to be installed on a system simply by visiting that site. However, later versions, see Figure 2, were spread using bogus downloads, email or document exploits; the latter becoming a more common attack vector as document applications become increasingly popular and complex, and therefore easier to exploit.

As with Polip, Kibik looks for available unused segments in the explorer.exe file. It is more careful than Polip in that there is no change in file size between the original and the modified versions.

When explorer.exe is restarted or the system is rebooted, the rogue code is used. When run, the modified explorer injects a thread from the virus's DLL (kibik.dll) into the following running processes, where available:

explorer.exe
iexplorer.exe
firefox.exe
opera.exe
avp.exe

These processes will usually have permissions to access the internet, which allows the Kibik to bypass desktop firewall policies.

Kibik performs a search on Google Blog Search using a hardcoded unique string. This would allow the malware to find and download additional instructions or malware. It is worth noting that this virus uses Google Blog Search which can link to blog sites via RSS or Atom feeds, giving it a flexible command and control channel.

It also contacts a CGI script hosted on the digiwexonline.com domain, which allows both an alternative form of command and control, and also lets the originator simply track the locations and number of infections.

Kibik's presence can be detected by:

- Certain files in specified directories
- Modifications to explorer.exe.

- Unexpected HTTP connections to the URL or blog cited above
- Modification to known Windows registry keys.

Grum-A worm

In the first quarter of 2007, there were a large number of emails that posed as messages from admin@microsoft.com, featuring subject lines such as "Internet Explorer 7 Downloads". These emails displayed a very believable image that invited readers to download beta 2 of Internet Explorer 7. Those who did click on the image downloaded a file called ie7.0.exe and were infected by the Grum-A worm.

Grum-A starts by inspecting the registry of a Windows system and looking for file names, which it then renames with a .rgn extension before beginning its infection. If the renaming is successful, then it creates an infected copy. It disables debugging privileges from the system so that some rootkit detection software will not initialise. It creates the following registry key to ensure auto restart on reboot.

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run:

Firewall auto setup (value) - %temp%\winlogon.exe

It hooks into selected APIs in a kernel library called ntdll.dll to hide itself. The ntdll dynamic linked library is the file that contains NT kernel functions. If these can be modified, the worm can hide itself. This can be done by replacing the standard preamble of selected APIs within ntdll.dll so that it jumps to the hidden code. The worm then needs to set up this hidden code in a suitably hidden location. It does this by using the executable file that it has renamed. It inserts the malicious code into the chosen file and copies it back to its original name. Finally, it gets commands and control from 72.232.49.214 for updates and instructions.

Fortunately, this gives us three ways of detecting its presence:

- The registry key used to ensure it can auto restart on reboot.
- The network activity involving IP address 72.232.49.214.
- The presence of .rgn files.

Why parasitic malware?

In the past few years, the game has changed for malware writers. It used to be for kudos and acclaim, or for revenge. Now it is for money. In the past, it was good to be discovered; now it is necessary to be hidden and difficult to detect.

From the above examples it is clear that the intention is to make detection as difficult as possible; to try and ensure that command and control channels are obscure and appear harmless; and, where possible, to ensure that infection goes unnoticed.

Visiting a website or downloading files from a P2P network are common and usually innocuous tasks. Parasitic malware fits perfectly into this new requirement. With the advent of rootkits to help hide the infection, it greatly improves the technique. One thing that is difficult for the blackhat, however, is having to do some sophisticated coding, such as moving segments in code to make space for their code, hooking into function calls and understanding kernel dynamic linked libraries.

"Visiting a website or downloading files from a P2P network are common and usually innocuous tasks. Parasitic malware fits perfectly into this new requirement"

The three examples given in this article demonstrate the care that has been taken to hide infection. No longer are we witnessing a big fanfare with something dramatic happening on screen. Instead, we have the likes of Grum, with its sophisticated and plausible graphic; Polip, with its seemingly-innocuous P2P file sharing activity; and Kibik, using a drive-by download exploit that most users would be unaware of.

Figure 3: Realistic malware graphic.

All of these try to hide themselves using the lessons learnt from rootkits, while the use of polymorphic features and the disabling of debug and emulation shows the lengths to which these trojans will go to in order to avoid detection.

Stealthy parasites

The Kibik virus is a particularly good example of how the new philosophy works. It used a vulnerability quickly and efficiently to install itself on numerous systems, the only indication being that IE may have hung or taken a lot longer to return when visiting an infected site. To the vast majority of users this would not be unexpected behaviour. Kibik was clever enough to infect IE, an application that is expected to access the internet and would not cause a lot of concern for being 'updated' or 'changed' as this too is commonly expected. Its command and control was through its access to Google Blogs, which might be unusual to some users if they spotted it but not to IDS or anti-virus software. The code was written to hide itself in IE without a notable difference in size.

It is evident from the above examples that many types of files lend themselves to parasitic malware. Ideally, the requirement is that there is enough empty space in a file that the blackhat can hide his/her code there. The technique is sufficiently mature that moving code and data segments

within a file are not obstacles, and this allows for larger trojans to be installed.

Detection

From the three examples given, it can be deduced that there are, typically, some symptoms of the trojan that can be detected, such as modified registry entries, new file types and increased files sizes. Grum's biggest weakness was the single IP address that was needed for command and control – removing access to this IP address effectively decapitates the trojan and limits its effectiveness. Grum also dropped .rgn files, leaving evidence of its presence.

Polip's use of Gnutella was clever because it prevented simple decapitation, making it reliant on the use of Gnutella which then identified that Polip was installed on a system. Furthermore, the files that it did infect grew in size, making the infection easier to find.

"It is undoubtedly getting harder to detect malware and parasitic malware, but thankfully there is usually something that allows anti-malware writers to pick up the scent"

Kibik's use of IE was clever, as was the code which was small enough to hide in IE without being detected and there was no increase in size of the IE file. However, the hash, effectively the fingerprint of the IE, was changed by Kibik, which made it detectable. So too did the connection to a given CGI script and the use of such a particular string to the Google Blog.

It is undoubtedly getting harder to detect malware and parasitic malware, but thankfully there is usually something that allows anti-malware writers to pick up the scent. The ultimate goal of any blackhat will be to hide so comprehensively that the changes cannot be seen. By no means any easy task, but one that they will doggedly pursue.

About the author

Simon Heron is the managing director of managed security company Network Box (UK) Ltd. He has more than 16 years' experience in the IT industry, including eight years' experience in internet security. During this time he has developed and designed a range of technologies, including firewalls, anti-virus, LANs and WANs.

Heron has an MSc in Microprocessor Technology and Applications, and a BSc in Naval Architecture and Shipbuilding.

Selling security to top management

Dario Forte, CISM, CFE, founder and CEO of DFLabs Italy

The most complex and important relationship in the cyber world is that between a company's top management and security management. Until recently, security was viewed mainly as a question of image, something completely unrelated to real business. When business needs and security needs were brought into the same conversation, it was usually in a conflictual manner. However, the last ten years have seen an increasing number of security incidents and the enactment of more and more laws and regulations to address the issue. This has inevitably led to a gradually deepening awareness on the part of both private and public organisations that effective security measures are a fundamental requisite for business integrity and for compliance with the law.

Dario Forte

Although an overblown 'regulation mania' is cited as one of the causes for the delisting of many companies in various markets, there are very real data protection needs that have importance well beyond simple legal compliance. These needs demand balanced security governance, an objective that can only be achieved via a committed relationship between top management and the company's information security organs.

How can balanced security governance be achieved, and what are its benefits? Generally speaking, the most important factor is a strong and visible commitment by top management, followed by their cooperation and constant communication with security management. Collaboration between information security and top management is the only way to optimise the protection of critical information and to ensure the elimination of conflicts among the various components of security, including privacy and information security. The result is fuller