Realtime publishers
"Leading the Conversation"

The Essentials Series: Modern Malware Threats and Countermeasures

Uncovering Modern Malware's Technologies, Behaviors, and Practices

sponsored by
Sunbelt Software

by Greg Shields

Copyright Statement

Uncovering Modern Malware's Technologies, Behaviors, and Practices

It is one thing to understanding what malware is. Understanding how it infiltrates into your computer systems during an infection is quite another. The discussion on types of malware in the first article of this series is intended to help enlighten your understanding that malware arrives in multiple forms, each with its own mechanism of infection, ways of replicating, and payloads for accomplishing its mission. The list in the first article of this series provides insight into the types of malware you've likely found on systems or heard of through the media.

The intent of this second article is to expand beyond mere classifications of malware and focus specifically on the technologies, behaviors, and practices used by each to infect computers, hide themselves, and remain resident on computers as long as possible. As with the evolution of the malware industry, new versions of its software have grown to become significantly more intelligent in recent years. Understanding how malware behaviors have changed over time will help you realize the scope of today's modern malware landscape.

Early Attempts

In the beginning, Windows-based malware attempts could be considered relatively easy to spot with the naked eye. In comparison with the malware of today, early infectors often utilized their own individualized processes for running the activities desired by the malware creator. Having their own individual process made early malware attempts easy to detect through casually browsing the contents of the Windows Task Manager. A skilled IT professional could search through Task Manager's list of running processes on an infected computer and often find processes that seemed out of place:

- Perhaps the process was not on the organization's list of those associated with approved applications
- Processes were sometimes masked with filenames similar to those already on the system, leaving duplicate entries in the list
- Occasionally, processes were given similar though not exact names, also making them stand out to the trained eye

In any of these cases, the process of determining that a computer was actually infected was an easier process than it is today. Although the removal of the malware components could be complex, its identification could be done through traditional IT troubleshooting techniques. Applications that assisted with the identification process could be run on an as-needed basis once the infection was confirmed to remove the offending software.

Modern Trickery

Today's malware, however, has reached a level of sophistication at which its identification can no longer be completed with the naked eye. In most cases, neither can it be identified using the common troubleshooting applications traditionally kept in the IT technician's toolkit. Though the different categories of malware utilize different mechanisms of trickery to evade detection, some current common behaviors make specialized applications necessary to find and get rid of the bad software. The following sections take a look at a set of these behaviors.

> This list includes only a subset of known behaviors. With malware development in a constantly evolving state, used mechanisms are always being modified to evade detection. Effective anti-malware tools incorporate always-on and real-time or near-real time updates to ensure that clients are always scanning using up-to-date methods.

System File Patching and Process Infection

When the typical IT professional thinks of the term "patching," this usually describes the monthly process done to update operating systems (OSs) and applications with newly released vendor code. But the concept of patching is much more general than this definition. Any file—whether it be an OS file, one for an application, or a data file itself—can be "patched" or updated with new code.

In the malware environment, this patching process involves reverse engineering known-good system files and injecting customized code into the result that is specific to the malware software. This activity commonly occurs with system files such as executables or library files used for processing of certain activities. In any of these cases, consider the result. Executables and library files within the Windows OS are collections of functions that enable an activity to be processed on the system. By extending these functions to include desired malware behaviors, the "patched" system file can now operate both for its original use plus the functions needed by the malware software (see Figure 1).

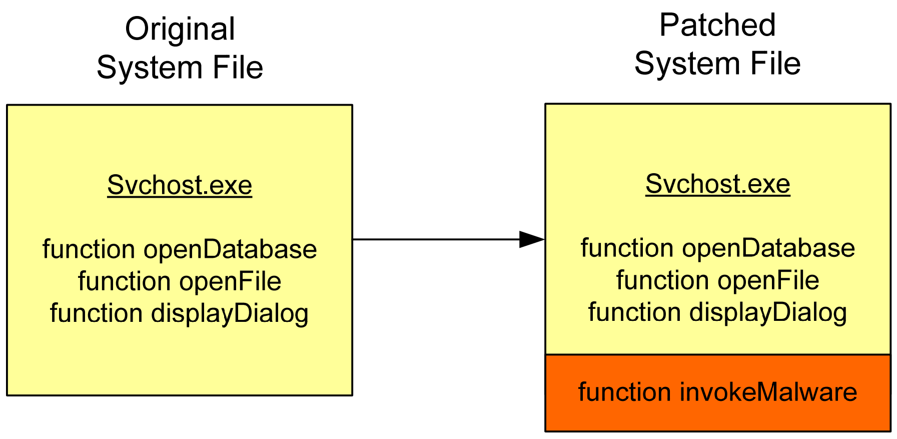

Figure 1: Once patched, the original system file can now perform malware functions.

The problem with identification in this case is that effectively nothing "new" has been added to the file system. The infected computer's list of running processes shows only expected processes, effectively hiding the malware's presence from the naked eye and common troubleshooting tools. The only way to eliminate the malware is to specifically target and remove the offending code snippets out of the file.

Code Resuscitation

Making this problem even more insidious is when multiple malware software packages are installed during a single infection event. When integrated during their code development, these packages can be augmented with the ability to monitor for the presence of each other. The resulting malware package can then monitor for the elimination of other malware packages resident on-system, similar to what Figure 2 shows. Should any of those packages be removed—typically as part of a removal activity—any remaining malware components still resident can restart or even reinstall the missing software from pieces left on-disk or downloaded from the Internet.

This assurance that malware remains resident on-system even during removal activities complicates the removal process. Standard removal tools such as manual file or registry deletions, process removal, or many of the other traditional troubleshooting tools available to IT technicians are insufficient to completely remove every piece simultaneously. Thus, missing even one component or not removing them all within a short enough time span results in the machine being re-infected within a short period of time.

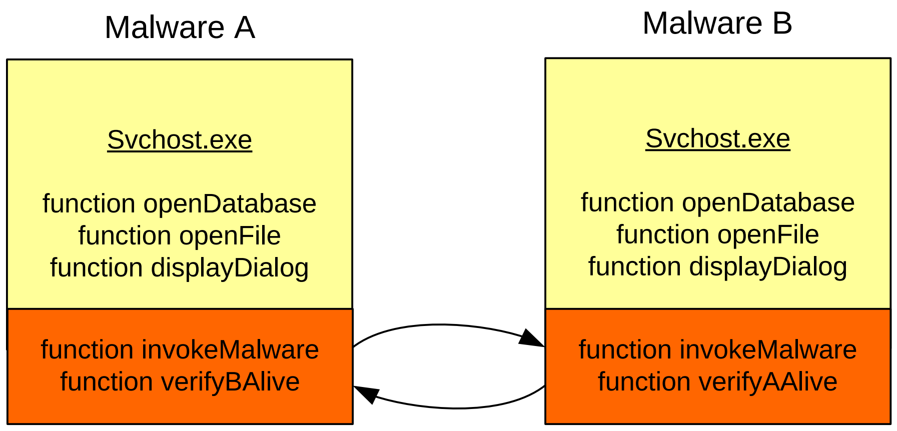

Figure 2: Malware instance A includes code that checks for the presence of Malware instance B. If not present on-system, instance A attempts to install or restart instance B.

Code Randomization

One mechanism for finding malware is to use code “signatures.” These tools heuristically look for the signs of malware code resident on an infected system. When the signature of a piece of code either in its own file or attached to a system file is found, the anti-malware tool can recognize its presence and attempt a removal. Signature-based detection is a common tool used with many anti-malware toolkits. However, the exclusive use of this method of detection is not without its intrinsic limitations.

Complicating the use of signature-based identification are randomization elements built-in to the self-compiling of the malware code as it installs itself. These features enable the malware to install itself onto a candidate computer using more than one possible configuration. Due to these features, the way the malware software “looks” to the detector actually changes over time. Figure 3 shows a simplistic example of how a function name can be slightly different in malware instance A than in its original form.

This level of randomization obviously requires enough similarity between instances so that its intended mission is accomplished. Yet, the randomization effects mean that multiple signatures must be created for a single malware instance as it replicates itself across the Internet. These code randomization features now seen in some of the most sophisticated of malware packages are a source for the dramatic increase in the number of perceived malware instances in today’s landscape. Only through equally sophisticated reverse engineering and behavior modeling on the part of anti-malware software and software companies can many of these malware “isomers” be identified.

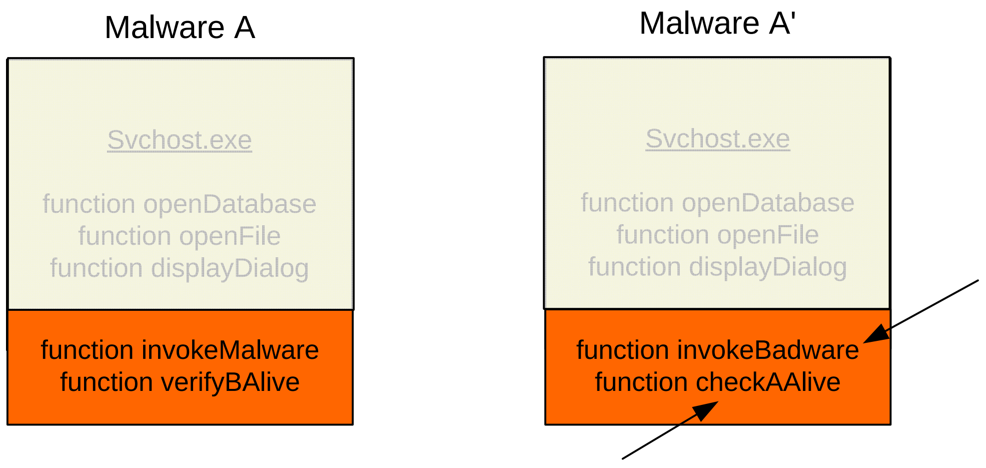

Figure 3: Subtle and randomized changes in the "look" of malware help keep it hidden from detection.

Rootkit and Cloaking Behavior

The last classification of modern behavior that is particularly dangerous due to its abilities to hide from even the native OS are rootkits. Although substantially different than the UNIX-based rootkits of yesteryear, this class of Windows malware focuses specifically on cloaking activities.

Rookits leverage some of the same file patching functionality described earlier but for a different purpose. Software patching in "regular" malware typically installs the types of functions desired for the processing of the malware's payload; rootkits are instead used to redirect system function calls through a process called "system call hooking."

Simply put, each system file is equipped with a set of addresses where it can expect to find other functions in system RAM. When it requires the use of another function, it seeks out that function at its listed address (see the top portion of Figure 4). In the case of a computer that has been infected by a rootkit, the rootkit modifies the target location for the other function to its own location in memory. This allows the rootkit to shim itself between layers of the system (see the bottom portion of Figure 4). There, it can identify what requests are being made by the system, and then alter the results of those requests as they are returned back to the function that initiated the request. Typically, the alteration is done to prevent the system from displaying the presence of files related to the rootkit itself or other malware on-system.

The end result of a rootkit infection is that the rootkit is effectively "invisible" to the system as well as to its user. Attempting to view the contents of a folder through either the GUI or command line is intercepted by the rootkit, and any trace of malware file presence is removed before the result is displayed to the user.

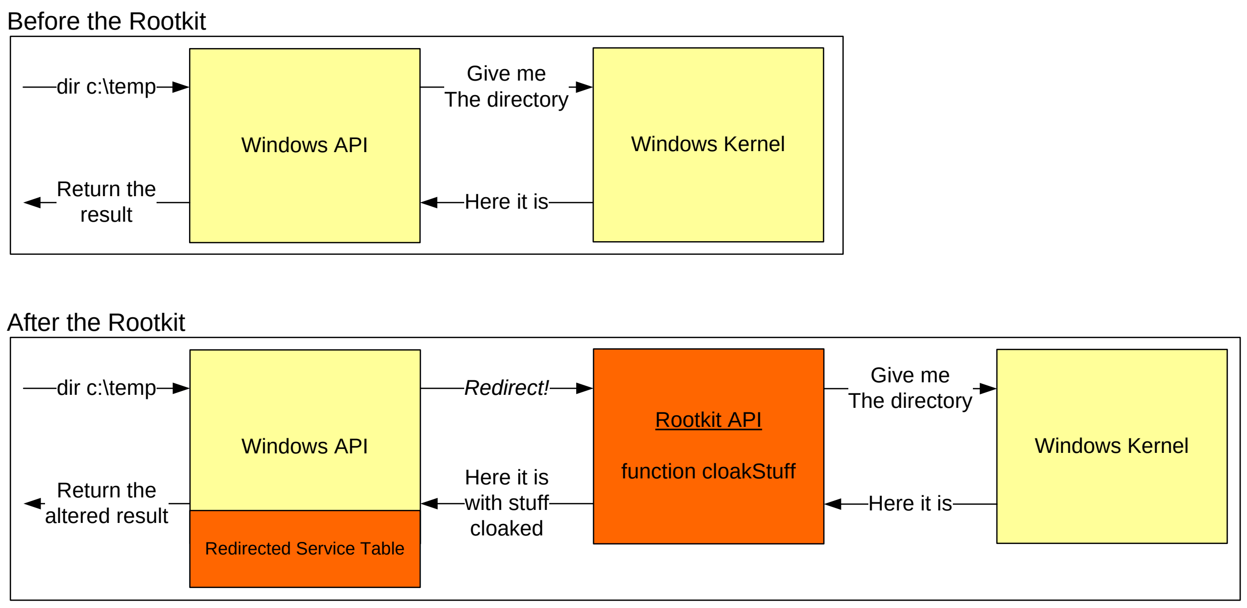

Figure 4: Upon installation, the rootkit shims itself between layers of the OS. There, it can view requests and alter associated responses without system recognition. The typical behavior is to cloak the presence of itself and other malware from the system.

Malware Is Getting "Badder"

When considering the economics associated with malware infections and the potential transfer of money as part of its development, it is easy to see why the level of sophistication is increasing over time. As the companies that write anti-malware software change their tactics, malware authors update their infection and replication mechanisms to suit. Only through effective software and diligent work in identifying these behaviors will IT organizations be able to maintain the health of their critical business systems.