SYMANTEC ADVANCED THREAT RESEARCH

The Impact of Malicious Code on Windows Vista

Orlando Padilla
Symantec Advanced Threat Research

Symantec Advanced Threat Research

The Impact of Malicious Code on Windows Vista

Contents

Abstract

This paper discusses the development of a framework to measure Windows Vista's immunity to legacy, current, and future threats. It presents metrics derived from this framework, security considerations applicable to Windows Vista, and a high-level view of what the future of malicious code may bring.

Introduction

As Microsoft unveils its new operating system, Windows Vista, it is important for us to consider the impact that its improved security will have on today's threats. A number of Vista's security improvements [1] were designed to reduce the impact of threats that users frequently encounter today. This paper describes a framework designed to determine the effectiveness of those improvements and to develop a baseline metric on how Windows Vista responds to different classes of threats. Emphasis was placed on measuring Windows Vista's immunity to threats by identifying successful execution of malicious code, the survivability of a threat after a system reboot, and circumstances where malicious code fails to execute. When examining the new security improvements in Windows Vista, we have found that the following technologies may have directly influenced the results:

- **User account control (UAC)** [2], originally termed least-privileged user account (LUA), was designed to force users running under administrative accounts to run as standard users, reducing the impact of malicious code. This feature restricts standard users from altering system-critical components transparently while allowing them regular usability.
- **Environment virtualization** refers to file and registry redirection. Virtualization in Windows Vista was primarily written to ensure the compatibility of legacy applications unaware of Windows Vista's new integrity level restrictions. A noncompliant application may require an administrative token to run correctly; Windows Vista deals with these legacy applications by redirecting a restricted write to a virtualized directory in the user's environment.
- **New firewall policies** now support bidirectional filtering and were developed to be application aware by default. Administrators can block an application from accessing local network resources without completely removing it or stopping execution.
- **Windows® Defender** is an antispyware technology adopted by Microsoft that works with Internet Explorer 7.0 to protect users from adware and spyware threats. It provides Internet Explorer with the ability to scan files before downloading them, identify phishing attempts, and protect system wide entry points for the installation of malicious code.

To test the effectiveness of these new technologies, we executed a subset of today's malicious code on Windows Vista. The classes of malicious code tested included backdoors, keyloggers, mass mailers, rootkits, spyware, adware, and Trojan horses. This paper provides an analysis of each threat class in order to determine its success. We also attempted to determine to what extent a threat is successful, answering whether it executed properly, whether it was able to survive a system restart, or whether it failed.

The paper is divided into two main sections: The first takes a detailed look at the framework's development and deployment, and the second examines the framework's results. It presents criteria supporting an implementation weakness in the Windows Vista default environment as well as information identifying the most effective security enhancements. Finally, the paper gives an analysis of the results and discusses how attackers will likely respond to the changes in the Windows security environment.

The framework

This section discusses the framework developed by Symantec for this project. The goal of the framework was to automate the execution of malicious code while monitoring its characteristics on Windows Vista. The framework includes three components, which are highly modular and portable. Following were the requirements for the development effort:

- Default Windows Vista installation
- All threats to be executed within the restricted user environment
- All tests to be repeatable

UAC was developed to minimize the attack surface of the operating system by forcing users to run in a standard (restricted) environment. Administrators, by default, are logged on to a standard user environment, and the operating system will either deny or prompt them for credentials when privileges are needed for a specific task. When the operation is denied, the task is not performed, and in the event of a prompt, users must provide the administrative password to elevate their privilege. This privilege separation effectively reduces the amount of damage the malicious code can do on a system. To handle applications not written with the Windows Vista new least privilege concept in mind, Microsoft virtualizes directory and registry I/O requests outside the user's environment by redirecting them to special folders and registry hives.

During the development of the framework, it was assumed that the majority of the consumer population would not alter UAC policy; therefore, threats would either have to evolve and utilize this new and restricted environment or find a way to elevate their privileges to carry out their tasks. For this reason, all tests were conducted in an environment governed by UAC.

Another important aspect of the environment for this testing was that all tests were executed under a VMware virtual machine. Note that some malicious code may intentionally not run under this environment. Several classifications were expected to fail but were included for the sake of completeness. In particular, because of UAC, rootkits, backdoors, and Trojan horses will inevitably fail, as by design they load drivers or attempt to modify systemwide settings. These threats will function if they are able to elevate their privilege to that of full Administrator. Additional research, outside the scope of this paper, has concluded that bypassing UAC and obtaining this required privilege level is possible, and not as difficult as originally expected.

MonServ

MonServ handles the monitoring plug-ins' instances for the framework (see the top left command shell in Figure 1). The monitoring plug-ins record the behavior of the system during the life span of the malicious code. Their main goal is to record pertinent system activity such as modified system settings, registry keys, newly created files, virtualized calls, and network activity.

MonServ is a simple server that listens on a network and takes one of three text-based commands:

- Start plug-in name
- Stop plug-in name
- Quit

The first two commands instruct the server to start or stop an instance of a monitor plug-in used to collect information about the system's current state. The last command instructs the server to stop all plug-ins and quit.

To function properly, MonServ must be started under a real administrator shell. This allows MonServ to load and unload drivers at runtime, which is required to successfully monitor I/O from kernel mode on the operating system. The plug-ins used for this framework were Process Monitor [3] and a Symantec internal tool called Observe, which monitors network and file system I/O. An instance of all the configured monitoring tools is loaded for each malicious binary executed to store independent log files.

Sovmain

The Sovmain component (see the top right command shell in Figure 1) is the core component of our framework. It loads and executes target malicious code binaries from a local directory and iterates through them sequentially. In each iteration, Sovmain sends a message to MonServ to start an instance of each monitoring plug-in available. Sovmain then launches the malicious code for an interval of 15 seconds and attempts to kill the process. This process (as indicated in the bottom right portion of Figure 1) then starts over again. Killing the malicious process results in unpredictable behavior due to the process's execution: It can crash, launch a separate process, or infect another file and exit. At this point, Sovmain instructs MonServ to stop monitoring and rotates the log files.

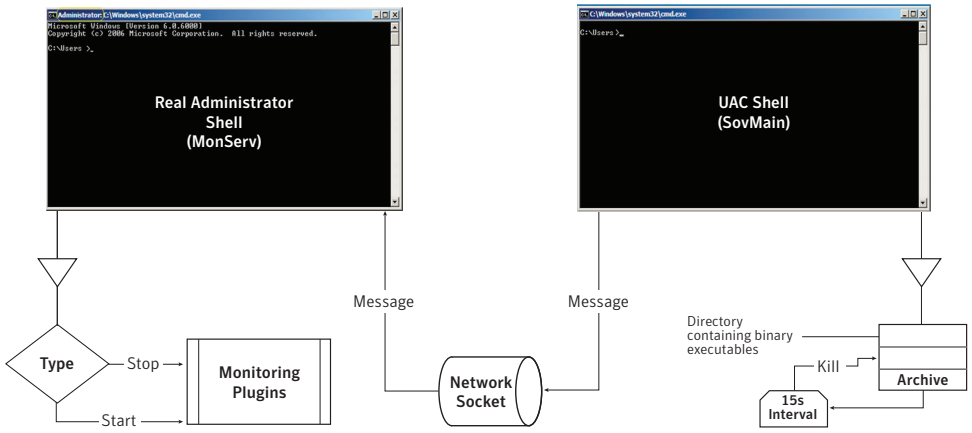

Figure 1. Program flow

Sovexamine

The final tool written for this framework is Sovexamine, which is used to normalize the large amounts of data recorded during each execution. The following is sample output from the tools used to evaluate the results of a given binary.

```
"11:39:55.874
AM","svchost.exe","852","FASTIO_RELEASE_FOR_CC_FLUSH","C:\$Recycle.Bin\
S-1-5-21-194839850-3226631451-796003310-1000\$I5LDJ2G.fs_dump","SUCCESS",""
"11:39:55.874
AM","svchost.exe","852","FASTIO_ACQUIRE_FOR_CC_FLUSH","C:\$Recycle.Bin\
S-1-5-21-194839850-3226631451-796003310-1000\$IY5HUHQ.hashes_dump","SUCCESS",""
...
"11:39:26.093 AM","WerFault.exe","2480","CloseFile",
"C:\Users\user\snakes on vista\malware\1da2_15b.ex__8376.exe","SUCCESS",""
"11:39:26.093 AM","WerFault.exe","2480","IRP_MJ_CLOSE",
"C:\Users\user\snakes on vista\malware\1da2_15b.ex__8376.exe","SUCCESS",""
...
"11/16/2006 - 16:21:9", 2972,
"\Device\HarddiskVolume1\Users\user\AppData\Local\Temp\",
"WERF1D8.tmp.appcompat.txt", "C:\Windows\system32\", "WerFault.exe", "11/16/2006
- 16:21:24", 988,
"\Device\HarddiskVolume1\Windows\System32\winevt\Logs\",
"Microsoft-Windows-UAC-FileVirtualization%4Operational.evtx", "<NO NAME>", "<NO
NAME>",
...
```

Logs formatted in CSV became increasingly difficult to read as they grew. Sovexamine took the data and carefully filtered out information that did not pertain to the analysis. The monitoring tools were not configured to exclude any information, as this may have led them to miss important characteristics of the malicious software. It was, however, trivial to identify most of the data that did not pertain to the testing. Post-filtered information included activity from the framework tools and Windows Vista default services (such as SearchIndexer, Lsass, and Windows Defender). After the exclusion process was completed, the data was organized into HTML tables, with certain criteria highlighted for quicker analysis. These criteria included, but were not limited to, system call return values; process names; virtualized calls; and, where present, system-handled exceptions captured by Werfault.exe.

Postexecution analysis

This section briefly describes the methods used to analyze the results of the framework's execution. The results, described in the following subsections, can be categorized according to three primary questions.

Does the malicious code begin execution successfully?

There are several caveats to be considered when labeling properly executed malicious code. A common example is the configuration of keyloggers; when executed for the first time, keyloggers often ask for certain properties (log file path, a password). These configuration windows are commonly displayed upon execution and usually function well, but when they are present, the keylogger will hang, keeping the configuration window open, or gracefully exit after failing to find configuration options. A second example relates to malicious code that immediately transfers execution to a spawned process and exits. With this in mind, the term "successful execution" used in the context of this paper means that the binary was mapped into memory by the windows loader and began executing instructions at the derived entry point. It does not, however, indicate that the malicious code successfully compromised the system's integrity.

Does the malicious code survive a system restart?

This category refers to any binary that successfully survives a system restart. A precompiled list of techniques that are commonly used to load executables at system startup was used to retrieve the total number of executables that use any one of the tested methods. The precompiled list, however, can become blurry if the malicious code latches on to a legitimate binary already in position to start at boot. Fortunately, since the framework itself monitors writes to the file system and registry, any successful write to an existing binary is accounted for. In the case of an unknown method of bypassing the Microsoft® Filter Manager [4], malicious code can evade the framework altogether.

Why does the malicious code fail?

This category refers to the analysis of the data collected during the project's lifetime. In Table 1, column 4, the column header "Failed Executions" refers to the number of executables that crashed at runtime—not executables that failed during the loading process or that were unsupported install packages.

An immediate indication of a binary failing to execute is the presence of the Windows Vista error-reporting process (WerFault.exe) during its execution time. WerFault.exe gathers information about the application that raised an unhandled exception. During this process, the faulting image is loaded, and the event is

logged by the monitoring tools. But one factor is currently not taken into account: If a process has its own registered exception handler or generates a dialog box as part of an attempt to recover from an exception, no WerFault.exe process is spawned.

Because of unmet conditions in the environment, some binaries followed a code path that exited with a negative value. As a result, monitoring the exit status of each sample was not an appropriate indicator of success or failure. This method can be prone to false positives, as nothing prevents a binary from exiting with an arbitrary value.

Table 1. Result statistics

Classifications	Execution Attempts	Executions	Failed Executions	Reboot Resilient
Backdoor	197	143 (72%)	24 (12%)	6 (3%)
Keylogger	118	60 (51%)	11 (9%)	5 (4%)
Rootkits	17	3 (17%)	7 (41%)	0 (0%)
Mass mailers	113	81 (71%)	7 (6%)	4 (4%)
Trojan horses	210	145 (69%)	24 (11%)	4 (2%)
Spyware	260	150 (58%)	24 (9%)	4 (2%)
Adware	118	74 (62%)	9 (8%)	2 (2%)
Unsorted	728	439 (60%)	103 (14%)	34 (5%)

Understanding the results

About 2,000 instances of malicious code were executed under the framework, including all classifications and a large number of uncategorized binaries. The binaries were extracted from Symantec's virus repository and attained by customer feeds or partnership programs with other antivirus vendors. They span a large area of malicious code and closely resemble what consumers deal with on a daily basis.

Table 1 illustrates the results produced by our framework and subsequent analysis. The remainder of this section details what the data represents. The table consists of five columns: Column 1 indicates the threat classification, column 2 contains the total number of binaries derived for this particular class, column 3 indicates the number of binaries that began execution successfully, column 4 indicates the number of binaries that failed to execute completely, and column 5 contains the number of binaries that successfully survived a reboot from the total number of execution attempts.

Backdoors

A total of 197 backdoor binaries were obtained for this project, of which 143 were executed successfully under the framework. Out of the successful executions, only 6 managed to modify a registry key or the startup directory, allowing them to execute on system start. Backdoors often rely on communicating with a remote host or binding a service to a port. Due to the default settings in the Windows Vista firewall, none of the binaries that executed under the framework fully compromised the victim host. The "Future Work" section describes an attack that enables attackers to bypass this default policy.

Keyloggers

Out of 197 keylogger binaries, 60 executed successfully, and only 5 managed to survive a restart. Often keyloggers come preconfigured to log to a default location that is usually in a directory other than that of the interactive user. This makes it difficult for a number of keyloggers, which fail upon execution due to lack of write permissions. If, however, an attacker is clever enough to change the configuration of a user-mode keylogger to log to a writable directory, the chances of it executing successfully are much greater. The successful execution count in the table indicates the number of keyloggers that did not crash when executed. The count, however, has no true association with the number of keyloggers capable of compromising the integrity of a victim's host. In fact, there are keyloggers that do execute successfully in the Windows Vista restricted environment. The "Future Work" section of this article describes how attackers can successfully load function hooks into a restricted user's environment, enabling them to log keystrokes.

Rootkits

As Table 1 illustrates, of 17 rootkits tested, none were successful at surviving a reboot, and only 3 executed successfully. Although a number of user-mode rootkits could potentially leverage local hooking methods to monitor interactive users (as explained previously), none succeeded under Windows Vista due to UAC restrictions. Kernel-mode rootkits fail immediately due to their inability to install drivers or use alternate methods, such as Direct Kernel Object Manipulation (DKOM)[5], for kernel infection.

Mass mailers

Of 113 mass mailers, 81 executed successfully, and only 4 managed to survive a system reboot.

The mass mailer category contained a few viruses such as Rontokbro.B [6] and variants that could potentially render a system unusable to a common user. Rontokbro.B is a highly aggressive virus: If any of a dozen or so keystroke sequences are detected or found in a window, the virus restarts the machine.

Trojan horses

The Windows Vista default policy settings played a key role in preventing many Trojan horses from performing network operations on the victim host. Of 210 binaries, a total of 145 executed successfully, and only 4 survived a reboot.

Spyware and adware

A total of 150 spyware and 74 adware binaries of 260 and 118 respectively managed to execute successfully. Only 4 spyware and 2 adware managed to survive system restarts. These two categories had a significantly higher number of failed executions, most of which related to missing Visual Basic runtime libraries and OCX components. The appendix contains a list of the OCX binaries that were included in previous versions of Windows but that are excluded on a default Windows Vista installation. The other executions that failed were typically caught by a UAC security restriction.

Unsorted

Unsorted binaries are of no particular class and were selected randomly from an unsorted repository. The unsorted list was used to broaden the number and class of binaries tested during this project. Out of 728 binaries, only 439 executed successfully, and 34 survived a reboot.

The future of malicious code

An analysis of today's malicious software provides insight into future methods of subverting Windows Vista security restrictions. Several key security enhancements provided by Microsoft successfully hinder much of the malicious code tested in our framework.

Although most of the restrictions are related to UAC, one in particular is not: The Windows Vista firewall is configured to disallow all network communications unless the user clicks the Unblock button. This feature, if slightly enhanced, would significantly limit malicious code looking to backdoor a host, including any third-party application trying to connect to a remote host or bind to a port to listen for remote connections. Unfortunately, the unblock function can be accessed with the same set of privileges as those of the restricted user. This configuration of privileges creates a point of vulnerability that undermines the effectiveness of the firewall's policy in Windows Vista. Malicious code can automate the unblock process by simply sending a message to the firewall pop-up dialog box via the SendMessage application programming interface (API) call.

UAC's restricted user environment posed the greatest obstacle for malicious code in the framework. The restriction of certain registry operations and writes to system directories as well as the design of consent are key features that diminish the likelihood of a successful attack or execution.

As we have observed however, a small percentage of malicious code threats are able to remain resident on a Windows Vista computer subsequent to a system restart. They are able to do so by leveraging the local user-owned versions registry keys that invoke applications during a system restart.

These user-owned keys are normally used to allow legitimate programs to load during the operating system's boot process. Attackers have used these same load points for malicious code in previous versions of Windows. Code attempting to execute during system initialization will need to be stealthier, as these keys are commonly monitored for new entries or modifications by antispyware applications such as Microsoft's Windows Defender. Previous attempts at hiding the modification of such keys depended on "blending" into the operating system's environment. Although this method may still be applicable under certain conditions, it is likely to fail due to the restricted account's inability to write to %SystemRoot%.

A more invasive but effective method would require a malicious application to identify third-party software that satisfies one of the following conditions:

- Contains an executable that is writable by the restricted user and uses one of the registry keys listed previously to launch during system startup.
- A nonwritable registry key points to a writable executable and is launched during system startup.

When the registry key is writable and the binary is not, the attacker can modify the key to point to an executable of choice and spawn the original process before executing its malicious payload. When the registry key is not writable but the binary is, the attacker can infect the third-party executable and perform an operation similar to that described previously. The startup directory in the user's default environment is another possible attack vector that can be leveraged to hide the execution of malicious code. Although these attacks are not systemwide, they should still be considered highly effective. The following sections describe the effectiveness of a nonadministrative compromise on Windows Vista.

Loss of data integrity

Assuming an attacker has used one of the methods described in the previous section to stay resident in the user's environment, the following nuisances are possible. An attacker can kill specific windows in use, reboot the system at will, or steal/delete/modify sensitive information—all of which may result in data loss.

Botnet capabilities

Assuming an attacker can perform the firewall unblock attack described in "The future of malicious code" section, most of the functionality commonly present in a bot is available. The functions available in a botnet include, but are not limited to, the following:

- Remote communication with the herder
- Scavenging of the local file system
- Participation in a distributed denial–of-service attack
- Network proxy
- Network reconnaissance
- Internet Explorer pop-ups

Eavesdropping

The API call SetWindowsHookEx [7] available in earlier versions of Windows allows applications to hook other API calls at runtime by injecting a user-specified call into a hook chain. However, the attacker may only infect processes running at the same integrity level as the threat itself.

A second API call of interest to malicious code authors is GetAsyncKeyState [8]. This call is used in conjunction with DLL injection methods to monitor keystrokes and other interactive activity on a desktop. The same restriction applies to this method: The attacker can only impact applications running at the same integrity level as the threat.

Future work

In building this framework we have identified a number of limitations that will need to be addressed in future iterations. First, we currently do not have the ability to identify user-mode hooks, nor to stop malicious code from rebooting the computer. These features would have enhanced our results. Another equally important consideration would be segregating the virtual machine instance for the execution of each sample in order to avoid the pollution of our base operating system image by malicious code. A less critical detail would be eliminating virtual machine dependency.

Future work will continue to examine the success of malicious code on Windows Vista, recording the number of variants released after a new initial attack vector and the measures that Microsoft takes in reaction to these trends.

Conclusion

To measure the impact of malicious code on Windows Vista, Symantec developed a framework to execute and track the behavior of a large number of malicious code samples. The majority of this malicious code targets Windows XP and previous versions of Microsoft Windows. As a result it was found to be much less effective when executed on Windows Vista. As Windows Vista becomes widely deployed, we expect that attackers will adapt to its new environment. As we have shown, threats can already execute within the confines of Vista's more restricted environment. In addition, research conducted outside the scope of this paper has shown that threats may leverage techniques to elevate their privilege to that of a full Administrator. Although Microsoft recently deferred from calling it a security boundary [9], UAC is designed to provide another layer of protection. With access to sensitive information, networking capabilities, and the ability to survive system reboots, we expect that threats already have the required functionality in order to propagate in the Windows Vista environment today.

Understanding a technology's threat space is one of the most important aspects of a security analysis. The greatest challenge to the development of a successful security model is achieving a viable security and usability tradeoff. If an operating system is impenetrable but is consequently unusable, then the tradeoff has failed. Attackers always have the upper hand, as their tradeoff is zero: Their goal is to compromise a system—with little or no regard for other consequences.

References

1. Dorman, Scott. “Windows Vista: Kernel Changes—Has anybody seen Gina and what’s a UAC?,” http://geekswithblogs.net/sdorman/archive/2006/06/17/82204.aspx

2. Microsoft. “User Account Control Overview,” http://technet.microsoft.com/en-us/windowsvista/aa906021.aspx

3. Microsoft. “Process Monitor v1.01,” www.microsoft.com/technet/sysinternals/utilities/processmonitor.mspx

4. Microsoft. “File System Filter Drivers,” www.microsoft.com/whdc/driver/filterdrv/default.mspx

5. HBGary LLC. “Direct Kernel Object Manipulation,” www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

6. Symantec Corporation. “Symantec Security Response,” www.symantec.com/security_response/writeup.jsp?docid=2005-100313-3908-99

7. Microsoft. MSDN Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/hooks/hookreference/hookfunctions/setwindowshookex.asp

8. Microsoft. MSDN Library, http://msdn2.microsoft.com/en-us/library/ms646293.aspx

9. Microsoft. “Windows Vista User Account Control Internals,” http://www.microsoft.com/emea/itsshowtime/sessionh.aspx?videoid=360

Appendix: Windows XP binaries not included in Windows Vista

Dynamic Libraries	OCX	Third-Party Mix
MFC71.DLL	MSCOMCTL.OCX	TABCTL32.OCX
MFC71u.DLL		MSWINSCK.OCX
MSVBVM50.DLL		BASE64.DLL
COMCTL32.DLL		WPCAP.DLL
		CVWRT.DLL

About Symantec

Symantec is a global leader in infrastructure software, enabling businesses and consumers to have confidence in a connected world. The company helps customers protect their infrastructure, information, and interactions by delivering software and services that address risks to security, availability, compliance, and performance. Headquartered in Cupertino, Calif., Symantec has operations in 40 countries. More information is available at www.symantec.com.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Boulevard
Cupertino, CA 95014 USA
+1 (408) 517 8000
1 (800) 721 3934
www.symantec.com

02/07 12065949