ANALYSIS OF THE WINDOWS VISTA SECURITY MODEL AND THE IMPLICATIONS IN THE NIGERIAN MARKET

M.A. SHAFI'I and M. D. ABDULMALIK
DEPARTMENT OF MATHEMATICS/COMPUTER SCIENCE,FEDERAL UNIVERSITY OF TECHNOLOGY MINNA, NIGER STATE

ABSTRACT

This paper provides an in-depth technical assessment of the security improvements implemented in the new Microsoft Windows Vista (officially released February, 2007), focusing primarily on the areas of User Account Protection and User Interface Privilege Isolation. This paper discusses these features and touches on several of their shortcomings. It then demonstrates how it is possible to combine these attacks to gain full control over the machine from medium integrity level. We also looked at the implications in the fast growing Nigerian market.

Key Words— *File Virtualization, Registry Virtualization, Integrity Level, UAP, LUA, UIPI, Computer security, Windows Vista, Windows Resource Protection.*

INTRODUCTION

Windows Vista is a radical departure from prior versions of the Windows operating system. With its introduction, enhancements have been made to virtually all aspects of the Windows security model. The operating system comes in five different versions i.e. Home Bases, Business, Premium, Ultimate, etc (with a sixth, "Starter" edition designed for developing countries), but only Windows Vista Ultimate--the most expensive one--includes the maximum level of protection. These changes should decrease the ease by which the operating system can be compromised.

As this paper is primarily focused on changes between Vista and the preceding Operating Systems, the reader is expected to have familiarity with the traditional Windows security model—including general knowledge of Access Control Lists (ACLs), System ACLs (SACLs) versus Discretionary ACLs (DACLs), Security Identifiers (SIDs),

SCOPE OF WORK

This paper focuses on attacks against the Windows Vista security model from the perspective of malicious code. The scenario addressed in this paper is an out-of-the-box configuration that a typical user will see when presented with a new Windows Vista installation. In this configuration the user is a Protected Administrator (Microsoft, 2006) using Internet Explorer 7 to browse a malicious website that exploits vulnerability (Microsoft, 2005). This vulnerability inadvertently introduces malicious code running with low privileges on to the host. In this paper, we discuss a technique whereby a weakness in Windows Vista builds will allow this malicious code to gain full control over the machine, ultimately acquiring LocalSystem privileges.

LIMITATIONS

Malicious code that is already running with full LocalSystem privileges is outside of the scope of this paper, since the malicious code has roughly the same capabilities as it had in previous versions of Windows. This paper only discusses the elevation of privileges to LocalSystem. Kernel-mode root kits are also outside the scope of this paper. An assessment of Windows Vista kernel mode security and new Windows Vista TCP/IP network stack will be covered in a separate research papers.

USER ACCOUNT PROTECTION (UAP)

Windows Vista introduces a security feature, User Account Protection (UAP), which is also known as Least-Privilege User Accounts or Limited User Accounts (LUA). User Account Protection feature is that when programs write to protected areas of the file system and registry, these writes are actually stored in a separate area, maintained per user, called the Virtual Store. This is very similar to what is done on a Terminal Server, and in fact I wonder why the Virtual Store is stored in C:\Virtual Store rather than under each user's Documents and Settings folder as is done on Terminal Server. This means that when running without restriction, the user is capable of activities such as installing software, writing to HKEY_LOCAL_MACHINE, starting drivers, starting services, etc.

However, all processes launched by the Protected Administrator run with minimal privileges. When a Protected Administrator launches a program from the Start Menu, the program will run in a restricted context with a smaller subset of the privileges than the user actually possesses. If the program requires administrative privileges (i.e., it won't function properly without them), the Protected

Administrator can run the process unrestricted. By running the process unrestricted, the process inherits the full privileges of the user (referred to as *elevation*). A program will be run in an elevated state using one of the mechanisms discussed in Section 3.0 below. Whenever a program is to be elevated, a popup box will appear asking the user to approve or deny. It is also possible to use a standard user account -- a user account without administrator privileges, rather than a Protected Administrator account. Standard user accounts were available in Windows XP. Although Microsoft recommends the use of standard user account, the default behavior when installing Windows XP or Windows Vista is to create an administrator user account. To create a standard user account, the user must perform additional manual steps. Therefore, we will only cover the default Windows Vista user account behavior that a general user would encounter after installing Windows Vista. This is not meant to imply that there are no privilege escalation attacks possible from a standard user account; rather, we focused our attention toward the most likely user configuration (Microsoft, 2006).

MANDATORY INTEGRITY CONTROL (MIC)

Mandatory integrity control (referred to here as *integrity levels*) is a new feature added in Windows Vista. It is controlled by an Access Control Entry (ACE) in the System Access Control List (SACL) of a securable object (e.g., a file, process, registry key, etc.). Integrity levels can be enabled/disabled via the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System\EnableMIC. Amusingly, the integrity level is associated with SACLs A process cannot interact with another process that has a higher integrity level. So CreateRemoteThread, SetThreadContext, WriteProcessMemory, and related APIs will fail from a lower integrity process when used against a higher integrity process. This is meant to prevent privilege escalation attacks. However it is still possible for:

1. A higher integrity process to callCreateRemoteThread, SetThreadContext, WriteProcessMemory, etc. against a lower integrity process.
2. Processes of any integrity level to interact using inter-process communication (named pipes, etc.).
3. A lower integrity server to impersonate a higher integrity client using APIs such as ImpersonateNamedPipeClient, as long as the impersonation level of the client allows it.

The following table shows the integrity levels and their effective permissions:

Integrity Access Level	**System Privileges**
High	Administrative (can install files to the Program Files folder and write to sensitive registry areas like HKEY_LOCAL_MACHINE)
Medium	User (can create and modify files in the user's Documents folder and write to user-specific areas of the registry, such as HKEY_CURRENT_USER)
Low	Untrusted (can only write to low integrity locations, such as the Temporary Internet Files\Low folder or the HKEY_CURRENT_USER\Software\LowRegistry key)

The integrity access levels are governed by the following SACL ACEs:

Security Identifier (SID)	**Integrity Level**
S-1-16-16384	System Mandatory Level
S-1-16-12288	High Mandatory Level
S-1-16-8192	Medium Mandatory Level
S-1-16-4096	Low Mandatory Level

)

UI PRIVILEGE ISOLATION (UIPI)

Directly related to integrity levels is User Interface Privilege Isolation (UIPI), which was added to prevent privilege escalation attacks such as Shatter (Russinovich & Solomon, 2005). If a lower privileged process is able to send window messages (using the SendMessage and

PostMessage APIs) to a higher privileged process, the lower privileged process can cause arbitrary code execution in the context of the higher privileged process. To address this, in Vista it is no longer possible for a process of a lower integrity level to send window messages to a higher integrity process. This is enforced by the windowing and graphics subsystem known as USER (presumably within the system driver win32k.sys). Certain processes, such as uxss.exe (Microsoft User Experience Subsystem) and consent.exe (Consent UI for administrative applications) are two processes that have the UI Access Mandatory Level, because they need to interact with the desktop.

RESTRICTED PROCESS

UAP is synonymous with *restricted process*. A restricted process is one with a restricted token that has some of the user's privileges removed and certain SIDs marked as "deny only". Restricted processes are setup using the CreateRestrictedToken API.

A **restricted** process created with UAP enabled has a reduced set of privileges:

SeChangeNotifyPrivilege	enabled
SeTimeZonePrivilege	disabled
SeIncreaseWorkingSetPrivilege	disabled
SeUndockPrivilege	disabled
SeShutdownPrivilege	disabled

By contrast, an **unrestricted** process created by an administrator has a much larger set of privileges:

SeChangeNotifyPrivilege	enabled
SeSecurityPrivilege	disabled
SeBackupPrivilege	disabled
SeRestorePrivilege	disabled
SeSystemtimePrivilege	disabled
SeShutdownPrivilege	disabled
SeRemoteShutdownPrivilege	disabled
SeTakeOwnershipPrivilege	disabled
SeDebugPrivilege	disabled
SeSystemEnvironmentPrivilege	disabled
SeSystemProfilePrivilege	disabled
SeProfileSingleProcessPrivilege	disabled
SeIncreaseBasePriorityPrivilege	disabled
SeLoadDriverPrivilege	disabled
SeCreatePagefilePrivilege	disabled
SeIncreaseQuotaPrivilege	disabled
SeUndockPrivilege	disabled
SeManageVolumePrivilege	disabled
SeImpersonatePrivilege	enabled
SeCreateGlobalPrivilege	enabled
SeCreateSymbolicLinkPrivilege	disabled
SeIncreaseWorkingSetPrivilege	disabled
SeTimeZonePrivilege	disabled

If a privilege is disabled, it means it is ignored during access checks but can be enabled by the process. If a privilege is removed instead of disabled, as is the case for restricted processes, it cannot be enabled.

UNRESTRICTED PROCESSES (ELEVATION)

A process will be elevated under a few circumstances:

1. If the application is an installer (has the extension ".msi", matches a common installer like InstallShield, is named setup.exe, etc.).
2. Application Compatibility (Slashdot, 2006).

(a) If the application has an application compatibility entry in the registry under HKEY_CURRENT_USER\Software\Microsoft\WindowsNT\CurrentVersion\AppCompatFlags\Layers\<path_to_executable> with the value RUNASADMIN.

(b) AppCompat database entry (a file that ends with <application_name>.sdb) created with CompatAdmin.exe.

3. The application's manifest file (<appname>.exe.manifest) or resource (embedded within the executable) that contains requestedExecutionLevel of requireAdministrator.
4. Manually by the user right-clicking on the executable and selecting "Run Elevated..." in Windows Explorer.
5. Also when a program is launched from an already privileged process.

THE LEGACY SHELL TRICK

WinLogon runs unrestricted and with high integrity. An executable that is launched from here via the "File -> New Task" menu option of Task Manager runs with full privileges. This does not seem to have been intentional. "The only way for the Shell to run as administrator is to log on with the machine administrator account (Built-in Administrator)." This statement is inaccurate. It was possible to kill the existing Explorer.exe from Task Manager and restart it via "File -> New Task" menu option, and entering "explorer". Task Manager launches processes via CreateProcess instead of CreateRestrictedProcess, so Windows Explorer is launched without restrictions and operates like the legacy shell from Windows XP. That is, there will no longer be any consent prompts when launching applications, and files can be moved, renamed, deleted without needing to elevate (Newsham, 2007).

FILE AND REGISTRY VIRTUALIZATION

Microsoft has introduced file and registry virtualization to retain applications backwards compatibility. When lower privileged processes that attempt to modify global locations fail due to lack of permission, the data is instead transparently written to a per-user location (known as *virtualization*). These per-user locations are checked before global locations. In other words, the per-user location overrides the global location (Newsham, 2007).

REGISTRY VIRTUALIZATION

Registry virtualization is implemented by ntoskrnl and ntkrnlpa (i.e., the operating system kernel itself). When running under a LUA process, registry write attempts that fail (due to insufficient permission) have their location changed from: HKEY_LOCAL_MACHINE\Software to: HKEY_CURRENT_USER\Software\Classes\VirtualStore\MACHINE\Software.

FILE VIRTUALIZATION

File virtualization is implemented by the file system filter driver luafv.sys. When running under a LUA process, file write attempts that fail (due to insufficient permission) have their location changed from: C:\Progra~1(C:\Program Files) to %UserProfile%\AppData\Local\VirtualStore\C\Progra~1

For example, if a LUA process tries to replace configuration file (e.g., %WinDir%\win.ini) and lacks sufficient privileges to modify the real %WinDir%\win.ini, then win.ini is virtualized to the per-user location. If that user later reads from %WinDir%\win.ini, the user will see his/her modifications (Newsham, 2007). However, no other users will see these modifications.

FLAWS AND ATTACKS

Windows Vista has unique challenges in trying to prevent privilege escalation attacks compared to the approach taken by UNIX derivatives. This section will focus on the privilege escalation attacks that result from the approach Microsoft has taken with Windows Vista.

If a higher integrity process uses registry keys and configuration files that are writable by a lower integrity process, then the security model is tainted, as this permits a lower integrity process to influence the behavior of a higher integrity process. In this section, we will show a number of flaws in the LUA implementation and, as a result, show how it allows privilege escalation from medium integrity, then medium integrity to high integrity, then high integrity to LocalSystem.

UNIX SECURITY MODEL

UNIX has supported standard user accounts with limited privileges for decades. Therefore, UNIX programmers are well adjusted to accommodating standard users. If a limited user wants to install a program into a global location that the user doesn't have write access to, the user will need to *su* (the switch user command) to a user that has write access to the global location (usually the root account). The limited user accounts also serve as a form of sandboxing. For example, it is common for the Apache web server to run under the user account apache. File permissions can be set to prevent apache from reading/writing to anything the web server doesn't need access to. Then if the web server is compromised, the attacker is restricted by the limited access of the apache account. We have over-

generalized this to avoid having to discuss specifics (Linux Security Modules, chroot, etc.). Some UNIX security models are quite similar to Windows Vista. For example, SELinux also has mandatory access controls and per-process privilege levels that are fixed at the time of program execution.

WINDOWS VISA SECURITY MODEL

Windows Vista's developers had to choose the best way to improve the overall security model while still retaining the most backward compatibility. While most of their decisions seem reasonable, two particular decisions lead to several seemingly intractable implementation flaws.

First, Windows programmers have been quite lax on leveraging and exercising rights and privileges in the existing Windows security model. A common behavior is to open a registry key or file for all access, when really only read access was needed. Another problem is making the assumption the user has administrative privileges (and requiring more privileges than actually needed). These assumptions are not just made by third-party Windows programmers—several Microsoft-implemented programs also fail without administrative privileges (e.g., the clock in the taskbar and shutdown.exe). For this reason, Microsoft has been forced to use "Application Compatibility" shims and file/registry virtualization to allow pre-Vista programs to function properly.

Second, Windows Vista can have several processes created by the same user operating at different integrity levels. This obviously creates an incentive for a low integrity level process to try to acquire the higher integrity level of the other process created by the same user. Windows Vista tries to close to obvious holes: for example, UIPI to prevent a lower integrity process from sending window messages to a higher integrity process. There is still far too much overlap between processes running at different integrity levels. A medium integrity processes can modify registry keys under HKEY_CURRENT_USER which are also used by high integrity processes.

CASE I: FROM MEDIUM INTEGRITY LEVEL

The low integrity escalation vulnerability was used to place "malicious.exe" in the user's Startup folder. Programs in this folder are executed when the user logs on to the machine. So for this section, it is assumed the user logged off and later logged back into the machine, resulting in the execution of "malicious.exe". Thus "malicious.exe" is now executing at the medium integrity level. To elevate privileges from medium to high integrity level, it is necessary to find a high integrity process that can be influenced by a medium integrity process. Possible attacks are:

- Find a shared memory section used by a high integrity process that is writable with medium integrity level (Russinovich & Solomon, 2005).
- Find a configuration file or registry key that is writable from a medium integrity process and used as input in a high integrity process.

CASE II: AGAINST WINDOWS RESOURCE PROTECTION

The following attack could be done from either LocalSystem or any account in the Administrators group, as long as it is a non-LUA process (since the SeTakeOwnership privilege is needed). LocalSystem and Administrators both have the ability to take ownership of files. Windows Resource Protection, as mentioned previously, is implemented as an ACL that only grants write access to the TrustedInstaller SID. However, because Administrators and LocalSystem both have sufficient privilege to take ownership of securable objects, the steps to evade WRP are to first enable the SeTakeOwnership privilege, second take ownership of the WRP-protected file or registry key, and finally grant Administrators full access. These steps can be done using the AdjustTokenPrivileges and SetNamedSecurityInfo. APIs. After that, the WRP-protected file or registry key can be changed without inhibition. There is no longer a thread that attempts to detect changes to protected system files as was done by SFP prior to Windows Vista. Therefore, it is possible to backdoor all system files at this stage. In addition, driver signing restrictions will not help to mitigate this attack.

FAILED ATTACKS

This section includes attacks that were unsuccessful. In some cases, the attack scenario was thoroughly tested and Windows Vista seems to properly defend against it.

Silent installs do not result in any silent elevation. Instead, a silent install runs with the credentials of the user and the install fails if more privileges are later required, without prompting the user.

Another method, that didn't work in testing, would be to place a malicious desktop.ini in a folder likely to be browsed Windows Explorer. Here is a sample desktop.ini:

```
[.ShellClassInfo]
IconFile=%SystemRoot%\system32\shell32.dll
IconIndex=-173
LocalizedResourceName=@shell32.dll,-12693
```

Windows Explorer checks for the presence of desktop.ini when browsing a folder to allow per-folder customization.

Attempts to override an existing executable such as %WinDir%\system32\calc.exe by placing a malicious calc.exe in the corresponding VirtualStore location failed. It was later determined that this is due to Windows Vista excluding certain file extensions(acm, cer, csh, hta, maf, maw, mst, pst, url, etc) from virtualization (Newsham, 2007). This seems to be undocumented in all of the Microsoft documents mentioning file virtualization, however this is clearly revealed by analyzing the file system filter driver that implements file virtualization (luafv.sys).

THE IMPLICATIONS IN THE NIGERIAN MARKET

The marketing propaganda touting Microsoft's new Vista operating system as "the most secure version of Windows yet" will not stop both white and black hat hackers in Nigeria and beyond from discovering Vista vulnerabilities.

Cost

Upgrading to Vista is very expensive, not only the new software but often new hardware as well. However, most Nigerians say that there is no reason to dump a functioning PC running Windows XP with Service Pack 2 and shell out $200(about ₦26,000) to upgrade to Vista.

Safety and Security

Safety and security is the overriding feature that most Nigerians will want to have Windows Vista for, if they are not into home entertainment or in any of the specialty areas, they are just going to feel safer and more secure by using it. Vista is light-years ahead of XP from a built-in security perspective. But Nigerians should know that *"No software is without flaws, and Microsoft will be the last to deny that".*

Internet Crime

The Economic and Financial Crime Commission (EFCC) have been doing a great job tring to cutdown internet crime in Nigeria. With the introduction of Windows Vista, the general expectation is that their work will be made much more easier. But we don't want people to expect that their computer is never going to be compromised becouse of Vista; that's simply not the case, the nature of maliciousness on the Internet is changing rapidly. It used to be that kids were trying to outdo other kids. But now, it is criminals! Ready to exploit the flaws in Vista.

CONCLUSION

In the face of known Vista security holes, Microsoft spokesmen have been unapologetic. Stephen Toulouse, senior product manager at Microsoft's "Trustworthy Computing Group," told CNN, "*We know from the outset that we won't get the software code 100 percent right ... but Windows Vista has multiple layers of defense.*" Another Microsoft representative told ZDNet, "*It's important to remember that no software is 100 percent secure.*"

Still, we wonder, "*Why is it important for us to remember that no software is 100 percent secure?*".

REFERENCES

Microsoft(2006). Access Control. *MSDN* [Online]. http://msdn.microsoft.com/library

Microsoft(2005). Developer Best Practices and Guidelines for Applications in a Least Privileged Environment. *MSDN* [Online]. http://msdn.microsoft.com/windowsvista

M. Russinovich, D. Solomon(2005). *Microsoft Windows Internals, Fourth Edition: Microsoft Windows™ 2003, Windows XP, and Windows 2000*. Redmond, WA: Microsoft Press, ch 8.

Slashdot(2006). First Windows Vista Security Update Released [Online]. http://it.slashdot.org/article.pl?sid=06/01/15/1910205

T. Newsham(2007). Windows Vista Networking: A Broad Overview [Online]. http://www.symantec.com/avcenter/reference