

User-Prompted Elevation of Unintended Code in Windows Vista

Overview

Windows Vista has implemented several new security features designed primarily to alert users to potentially-dangerous situations on their computers and prevent malicious software from accessing critical system components. One of the most-touted features by Microsoft, and perhaps the most visible security addition to Windows Vista is User Account Control (UAC), in which even computer administrators do not run with full administrative privileges. This guards the user from potentially-malicious software by preventing processes from writing to system folders, such as %SYSTEMROOT% and \Program Files, as well as writes to the portions of the registry that are not user-dependant, including the HKEY_LOAL_MACHINE (HKLM) and HKEY_CURRENT_CONFIG (HKCC) registry hives.

How UAC Functions

UAC, which is enabled by default, functions by “splitting” the user’s security token so that the “Administrator” role is not part of the user’s default security context (see Figure 1, right). When an administrator starts a process, the standard user access token is assigned unless the process is explicitly started as an administrator, either by right-clicking on the program (or its shortcut) and choosing “Run as Administrator,” (see Figure 3, below) or by designating in the program’s Compatibility tab in Windows explorer that the program requires administrative privileges. Doing so will cause

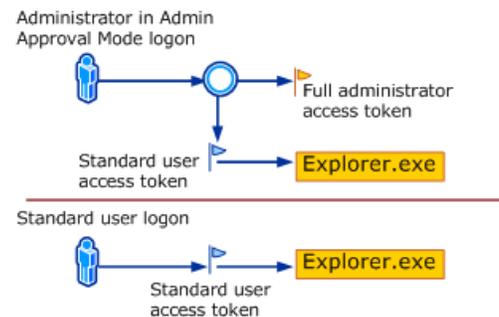


Figure 1: The Windows Vista UAC model. Image from <http://technet2.microsoft.com/WindowsVista/en/library/00d04415-2b2f-422c-b70e-b18ff918c2811033.msp?mfr=true>



Figure 2: A desktop icon that requires elevation to run.

Windows to prompt the user to “elevate” the process – for administrative users, this involves clicking a “continue” button; regular users, however, must enter the credentials for an administrative user account.

To improve security, Windows Vista provides a number of visible cues to indicate that a process requires elevation. One such cue is the Shield icon (see Figure 2, left) displayed on the bottom-right corner of any shortcut or executable that requires administrative

privileges. When activating a program that requires elevation, Windows grays out the desktop, switches to the

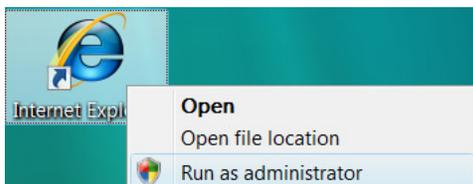


Figure 3: Elevation can also be done via a right-click.

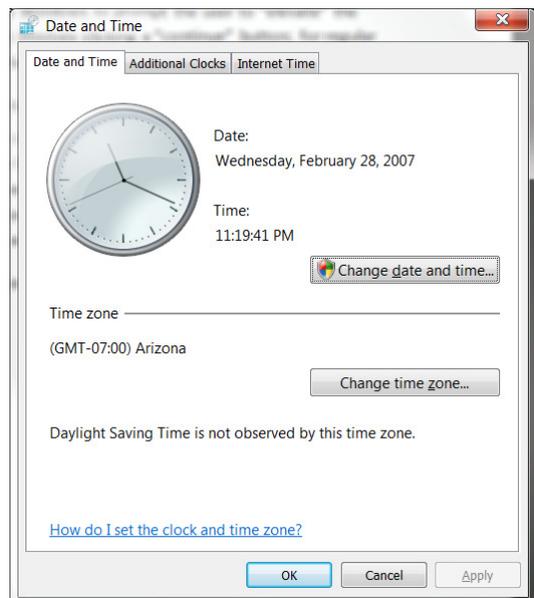


Figure 4: The standard Date and Time dialog does not allow setting the date, but you can set your time zone.

secure desktop, and prompts either for elevation (this is known as “Admin Approval Mode”), or prompts for administrative credentials when running as a standard user.

Also important to point out is that several Microsoft user interface applications are able to request elevation for dialog elements while not elevating an entire process; this is commonly seen within Windows Explorer, when performing writes to system-protected folders, or in Control Panel applications. One such example of this is the Date and Time dialog (see Figure 4, above) – users may set their time zones independently of other users, but administrative privileges are required to set the actual time and date. The Shield icon appears on the “Change date and time...” button, indicating that elevation of privileges are required to perform that task. This functionality is enabled through COM, as COM enables creation of objects out-of-process.

Additionally, once a process is elevated, any additional processes and threads spawned by that process will have the elevated security token. A critical example of the importance of this is the command prompt: when it is running with elevation, the dialog will prefix “Administrator:” to the caption (see Figure 5, right). If you

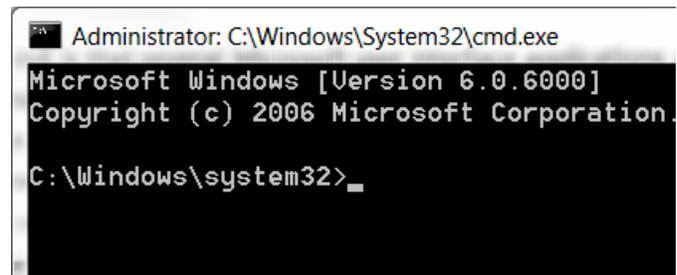


Figure 5: The command prompt displays “Administrator” in the title bar caption to indicate that it has elevated status.

attempt to start the Microsoft Management Console (mmc.exe), which requires elevation by default, you will find that you are not prompted for elevation. Other tasks that typically require elevation, including such activities as modifying %SYSTEMROOT%\System32\drivers\etc\hosts, are also permitted by the command prompt in elevated mode and any other processes started by the command prompt, even once the command prompt has been closed.

One last point to note about UAC is that Windows Vista uses a number of heuristics to determine whether a program should be automatically elevated, even if it is not marked for elevation by default. This most frequently is the case for any kind of setup file (*.msi, for Windows Installer packages), and any executable files that might have “install” or “setup” in the name. Windows prompts for elevation whenever it detects a setup process, although its heuristics are not always effective. For example, the installer for IrfanView 3.99, a free image-processing tool that supports a wide range of file formats, is shipped with a home-grown installer that does not request privilege elevation through either an assembly manifest and does not have the words “setup” or “install” in the file name (the default downloaded filename is iview399.exe). UAC redirects the setup program’s output to the user’s virtualized store, and upon completion, UAC notes that the program “may not have installed correctly,” prompting the user to restart the process with elevated privileges. This virtualization allows programs that write to system areas to function without breaking in Windows Vista under UAC.

To verify that the process should be elevated, Windows will gray out the screen and present one of several dialogs that indicate the level of trust you should have, and ask for appropriate confirmation to continue. The level of trust is determined by system policy and the type of application that is being run. Example dialogs appear in the following table:

Table 1: The four levels of UAC dialogs in Windows Vista

Code Level/Application Information	Sample Screenshot
<p>Application is blocked because the Authenticode-signed publisher is blocked by policy (the application will not be run).</p>	<p>The screenshot shows a User Account Control dialog with a red header bar. The title is "User Account Control". The main message is "This program has been blocked". Below this, it says "Your administrator set policy to block this program." and lists "SmartFTP.exe" as an "Untrusted Publisher". There is a "Details" link and a "Close" button. At the bottom, it says "User Account Control helps stop unauthorized changes to your computer."</p>
<p>The application publisher is Windows Vista, and may be part of the operating system (such as a Control Panel application).</p>	<p>The screenshot shows a User Account Control dialog with a blue header bar. The title is "User Account Control". The main message is "Windows needs your permission to continue". Below this, it says "If you started this action, continue." and lists "IIS Manager" as a "Microsoft Windows" application. There is a "Details" link and "Continue" and "Cancel" buttons. At the bottom, it says "User Account Control helps stop unauthorized changes to your computer."</p>
<p>The application publisher is Authenticode-signed, or the application is recognized as approved by system policy.</p>	<p>The screenshot shows a User Account Control dialog with a grey header bar. The title is "User Account Control". The main message is "A program needs your permission to continue". Below this, it says "If you started this program, continue." and lists "SmartFTP Client" as a "SmartSoft Ltd" application. There is a "Details" link and "Continue" and "Cancel" buttons. At the bottom, it says "User Account Control helps stop unauthorized changes to your computer."</p>
<p>The application is not Authenticode-signed, and the application is not recognized as approved by system policy.</p>	<p>The screenshot shows a User Account Control dialog with a yellow header bar. The title is "User Account Control". The main message is "An unidentified program wants access to your computer". Below this, it says "Don't run the program unless you know where it's from or you've used it before." and lists "Converter.exe" as an "Unidentified Publisher". There are two options: "Cancel" (with a right-pointing arrow) and "Allow" (with a right-pointing arrow). The "Cancel" option has the text "I don't know where this program is from or what it's for." and the "Allow" option has the text "I trust this program. I know where it's from or I've used it before." There is a "Details" link and "Continue" and "Cancel" buttons. At the bottom, it says "User Account Control helps stop unauthorized changes to your computer."</p>

These dialogs may be different if the system is prompting for credentials, but the basic style of the dialogs, the color, and the text content will be generally the same.

Attack Vector – High Level Overview

This vulnerability uses a two-step attack vector against a default installation of Windows Vista. Initially, a malicious software program can be downloaded and run without elevation, and this downloaded software program, called the **proxy infection tool**, can behave as expected while it sets up the second-level malicious payload. For instance, if users believe they are downloading a “Pac-Man” clone, such a game could be run while the malicious software did its work in the background). It is important to note that, realistically, once the proxy infection tool has been run on the target machine, the target is effectively infected. This pattern of infection follows the typical Trojan horse model, piggybacking on what may be otherwise legitimate software.

Before discussing what the proxy infection tool does, a brief overview of the construction of the Start menu is warranted. The Windows Vista Start menu is synthesized from a user’s Start Menu\Programs folder and the computer’s All Users\Start Menu\Programs folder. On a default installation, these folders are located at:

All Users: C:\ProgramData\Microsoft\Windows\Start Menu

Specific User’s: C:\Users*User Name*\AppData\Roaming\Microsoft\Windows\Start Menu

For the remainder of this document, the term `%ALLUSERS%` will refer to the above “All Users” file path, and `%USER%` will refer to the specific user’s file path. Also, we will label the following path `%USERDATA%`:

Specific User Data: C:\Users*User Name*\AppData

In Windows Vista, the two folders are merged to produce the “All Programs” list, and duplicate items have the specific user’s folder preferred over the all users folder. Thus, if a user has a shortcut file called “Paint.Ink” in `%ALLUSERS%\Accessories` and `%USER%\Accessories`, the shortcut in `%USER%\Accessories` will be displayed on the Start menu. This is the mechanism by which the user will raise a malicious program to elevated status.

Any user has permission to write to his or her `%USER%` folder, as well as the `%USERDATA%` folder. This allows a program to place arbitrary shortcuts on the Start menu. This is as simple as utilizing the `IShellLink` and `IPersistFile` COM interface to create, resolve, and modify shortcut (*.lnk) files. By replicating shortcuts found in the `%ALLUSERS%` folder into the `%USER%` folder, and redirecting the replaced files to point to alternative executables, malicious software can inject itself into the user’s Start Menu without requiring elevation. By intercepting processes in this way, the malicious software can monitor a user’s browsing habits, monitor process starts, and aggregate data until it has the opportunity to elevate itself to administrative privileges.

Attack Simplification Due to Software Included with Windows Vista

Because of a number of new technologies and programming APIs are included in Windows Vista that were not available by default in previous versions of Windows, the attack is significantly simplified. Specifically, the .NET Framework 2.0 CLR comes preinstalled on Windows Vista (whereas it was an optional update in XP) and provides a wide array of rich development tools with which to rapidly create software for any number of purposes. Because the .NET Framework has a rich class library, the actual malicious code footprint can be small; no large regular expressions libraries would be necessary, for example, nor would some cryptographic toolkits.

The code presented to exploit this vulnerability uses the .NET Framework's System.CodeDom namespace to generate stub applications that reside in the %USERDATA% folder. Shortcuts are created that point to stub applications, *but only for executables that are not signed*. The reason this is important is that, as noted before, signed and unsigned executables have different UAC confirmation dialogs. Since we will not have the benefit of being able to sign our malicious executables identically to the company that deployed the real application, we will not insert stub application shortcuts in the path of executables that are signed.

Attack Vector Low-Level Implementation

Workflow

Figure 6 (below) outlines the attack vector's low-level implementation workflow:

- 1.) The user downloads the Trojan horse onto the target machine. This is perhaps the most limiting part of the attack; however, given the proper motivation to execute unknown software, or perhaps untrusted software, this should not be a major barrier to infection. Common examples of this could be the millions of e-mail worms that have been downloaded, games, Internet Explorer add-ons, or any other common vector of Trojan horses.
- 2.) The user executes the software as a standard user. The Proxy Infection Tool has a variety of options: it can be part of the Trojan process, or it can be extracted as a resource and executed as an independent process. The latter is the preferred method of infection, as if part of the Trojan process, its work will be terminated when the parent process ends.
- 3.) The Proxy Infection Tool writes malicious code in the form of a DLL to the %USERDATA% folder and probes the All Users start menu folder, the user's specific start menu folder, and the user's Desktop and Quick Launch folders for shortcut (*.lnk) files that are candidates for replacement.
 - a. The PIT loads each shortcut and examines the target executable.
 - b. If the target executable is unsigned and is not part of the OS, proceed. Otherwise, skip the given shortcut and continue iterating.

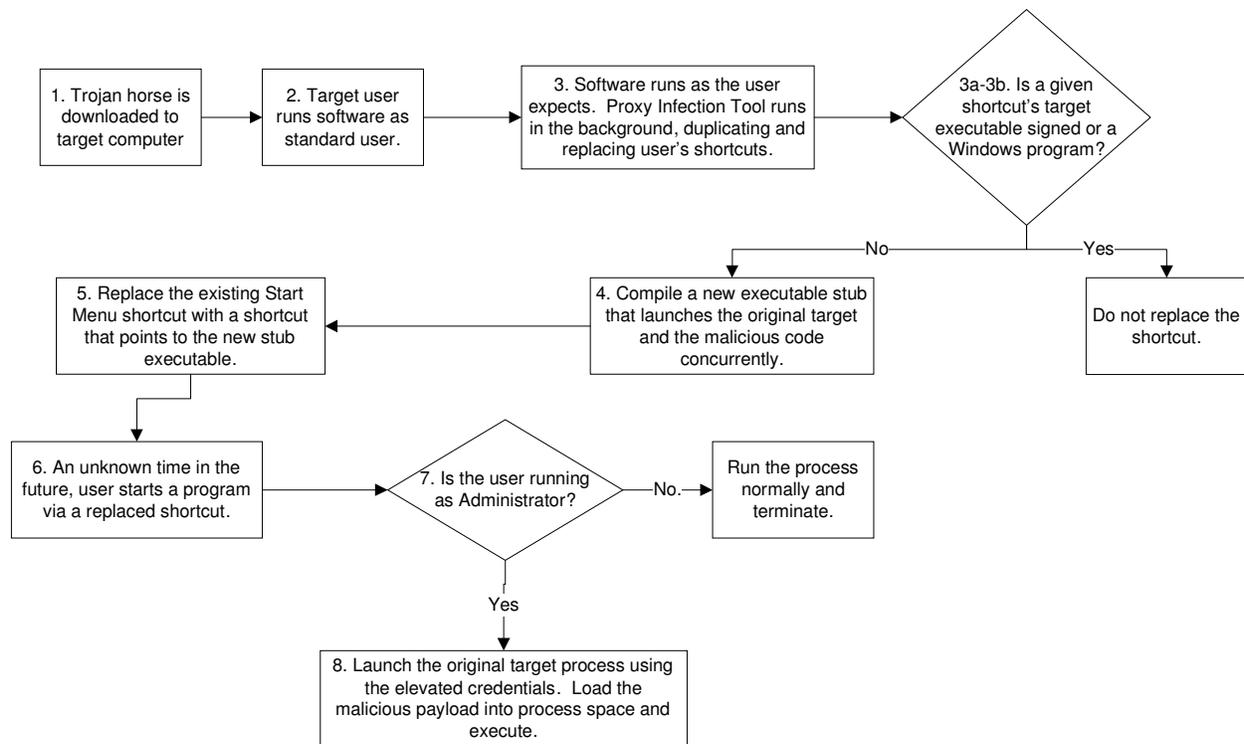


Figure 6: The generalized workflow diagram for infecting a target machine.

- 4.) Compile an executable stub to launch the original target executable and store this in a user data path (such as a descendant of %USERDATA%). The stub should automatically launch the original target. The stub can either be generated by string assembly/concatenation, or via the `System.CodeDom` utility classes. It can also be generated by an expert in assembly language generation and programming.
- 5.) Write the shortcut back to the user's Start Menu via the COM `IPersistFile` interface. The new shortcut should point to the newly-generated executable.
- 6.) At some time in the future, the user will start an intercepted shortcut with elevation. This may be done for a variety of reasons (see Attack Vector – High Level Overview, Potential Targets above for more information). The stub executable launches the original process with elevation.
- 7.) The stub executable checks to see whether it is running with elevation by determining whether the current Windows principal¹ is in the "Administrator" role.
- 8.) The malicious program is loaded into the process. This is likely done by loading a DLL from the %USERDATA% folder, which was written in step 3, and reflecting against it to load the malicious code's type information via metadata. This can be done in very little code in the code graph, and can have several different appearances (a candidate Type can be extracted from the DLL, the Type can be specified in the code graph, the Type can be specified by metadata, etc.). The original target process and the malicious process then run in parallel.

¹ In the Windows security model, a Principal is the security context of the user on whose behalf code is running, which is made up of the identity of the user (the user's logon account) and any roles (user groups) to which the user belongs.

Determining if the User is running as an Administrator

The inclusion of the .NET Framework makes it trivial to determine whether the current user is running as an administrator. Listing 1, below, demonstrates a console Windows program that prints a line to the console based on whether the user is running as an administrator or standard user.

Listing 1: A sample C# program that checks to see whether the user is running as an administrator.

```
using System;
using System.Security.Principal;

class Program
{
    static void Main(string[] args)
    {
        WindowsPrincipal wp = new WindowsPrincipal(
            WindowsIdentity.GetCurrent()
        );

        if (wp.IsInRole(WindowsBuiltInRole.Administrator))
        {
            Console.WriteLine(
                "Currently running as an administrator."
            );
        }
        else
        {
            Console.WriteLine(
                "Either running as a limited user or in admin-approval mode!"
            );
        }
        Console.ReadLine();
    }
}
```

When implemented, the generated stub executables would be built with the logic demonstrated above, and simply enough logic to load and execute code from an external DLL.

Attack Vector Defenses

There are several defenses with varying degrees of success that system administrators can take against this attack. However, with the wide availability of tools available online for such things as executable packing, and the given initial infection vector, as well as the wide array of low-footprint options available to the developer, even malicious software detection tools such as agent-based anti-virus software may find it difficult to find this software.

The best place to stop this attack would be at the Trojan horse level. If the proxy infection tool is not allowed to run, there will not be any malicious elevation in place. However, because it is likely that the proxy infection tool is going to be run, perhaps unwittingly, automated defensive systems should evaluate executing assemblies outside of what would be considered the “protected” system area.

Ideally, software should be blocked from running from user data paths unless otherwise unrestricted by policy, analogous to a no-execute (NX) bit² (although it could be done with NTFS permissions, such an interface is likely not intuitive for a majority of users). With the exception of developers, who may work on projects within their user folders and then run the built executables from there, rarely in a corporate environment is it desirable to have software running from the user's desktop or documents folder (as they likely downloaded it).

However, justifying this on a home user's machine is much more difficult, and may not be – and is arguably not – worth the trade-off of functionality vs. security, as users commonly download executables for a variety of reasons. Additionally, a risk is inherent in that installers can be delivered as executables that do not demand immediate elevation: when an executable setup program is run, but does not demand elevation immediately because it is not detected by heuristics, cancelling the setup program does not negate the risk of infection. One such example of this is the IrfanView installer, noted above.

Potential Uses for Attack Vector

Potential Targets

There are any number of potential target software packages that could be attacked by both the Proxy Infection Tool and the malicious code stub. There are also various levels of automatic stub generation: stubs can be generated only for applications that are marked as requiring elevation, or for all shortcuts, depending on the level of sophistication of the Proxy Infection Tool.

Targets that potentially may not be attacked

First, there is a subset of programs that simply may not be attacked: namely, Windows OS-signed executables (such as mspaint.exe), signed and trusted executables as defined by system policy, and signed but untrusted executables as defined by system policy. The desired elevation prompt when the program is normally run is the yellow prompt in Table 1 (above); this is because the user will not be able to typically distinguish between the stub executable and the actual target executable without expanding the “Details” pane. Because signed executables have Authenticode[®] signatures and the Proxy Infection Tool will not have the private key to sign generated stubs, changing the elevation prompt may alert the user to the infection, and so this is not the default behavior of the infection code. However, it is a trivial matter to modify the code to allow interception of such executables.

Potential targets for the Proxy Infection Tool's redirected shortcuts

The real danger of this attack is the wide variety of software that it targets as well as the sheer transparency of the attack to the end-user. There are a variety of scenarios in which the redirected shortcuts may be elevated and the yellow prompt displayed:

² An NX bit, or no-execute bit, is a hardware page table feature of newer processors that designates a memory page as not executable. When an attempt is made to execute code in such a region of memory, the processor raises a general protection fault. Windows Vista can also simulate this behavior using only software to some degree, through a feature called Data Execution Prevention (DEP).

- Amazingly, Microsoft did not sign the Internet Explorer 7 executable. Right-clicking on the Quick Launch bar's Internet Explorer icon and running it as an Administrator presents the yellow (unsigned) elevation dialog.
- Any target process that must write, for any reason, to the \Program Files folder. For example, *The World of Warcraft*, a game produced by Blizzard Entertainment, with an install base of roughly six million, is unable to persist settings, or more pressingly, patch itself, without such permission. Users must typically elevate this shortcut twice to successfully play. Since this game does not allow users to log on unless they are running the most recent version of a client, it is a prime target for the redirection attack.
- Any shortcut to a setup, or more commonly, uninstallation program. Windows uses heuristics to determine whether a program will require elevation, such as whether the program has "setup" or "install" in its name. Any such shortcut is an excellent target for the redirection attack.
- Any program that is marked for elevation that is unsigned. This will, of course, be automatically picked up if all shortcuts are replaced. However, if they are not, the list is set per user in HKCU\Software\Microsoft\WindowsNT\CurrentVersion\AppCompatFlags\Layers; any given key that specifies "RUNASADMIN" as a flag is a candidate for replacement. This flag is set when the user designates "Run this program as an administrator" on the Compatibility tab of the file or shortcut's property pages.

Once Infected

Finally, there are several methods with which to bypass UAC once the target has been infected and the process run as an administrator. In particular, once software is run as an administrator, it has full access to the system registry. From there, it can register itself to be run on startup under the HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce registry key; Windows Defender blocks software that requires elevation from running from other keys, but software in this key is treated as if it was part of an installer and is elevated automatically. Consequently, the process can add itself again to the RunOnce registry key and perpetually run itself as an administrator.

Conclusion

Circumventing Windows Vista's User Account Control feature is not the easiest task to complete. However, because of the user interface of Windows Vista, as well as the software platform of the .NET Framework integrated into default installations, malicious software is able to interject itself between the user interface and the software programs the user interface represents. Additionally, the code appears to be entirely benign to heuristic checks by malware scanners, and a generalized delivery system can implement any number of possible activities sought.

Windows Vista introduces a number of security enhancements designed to prevent many of the kinds of security issues seen in the past. Address Space Layout Randomization (ASLR) sets the base virtual address of executables to a random value, utilizing the relocation information found in Windows PE files to adjust pointers accordingly. ASLR hardens Windows particularly against buffer overflow attacks,

because it makes predicting the target of a jump instruction more difficult, making it significantly more likely that a program will crash. Windows Vista also performs stack hashing to detect buffer overflows, which causes a program crash should it find a problem. User Account Control, obviously, also protects the user from software in the event that malicious software should be executed, that it is unable to overwrite critical system files and settings.

Because of the new security features, exploits in the near future will likely depend more on user interaction that automated attacks against such targets as network devices and network stacks. Unfortunately, it is exceedingly difficult to effectively educate end users of the security implications of their actions. And, as has been a repeatedly-heard criticism of UAC, users will be conditioned to simply click the "Continue" button.

Unfortunately, relying on security software such as virus scanners or Windows Defender to prevent the exploitation of this vulnerability brings the user back to square one: User Access Control might as well not exist. Under the proper conditions – those that come with Windows Vista by default – UAC is rendered useless as a result of this vulnerability.

What can Microsoft do?

Unfortunately, because of the design of the Windows user interface, there is not an immediately-viable solution to this exploit. There are a number of solutions that Microsoft could implement, however, which would make it more apparent to the user if such an exploit was in use:

- A fifth UAC dialog box could be introduced, alerting the user if the shortcut was started from the user's Start Menu list rather than the All Users list, and if the executable in question is in a non-UAC-protected location.
- A fifth UAC dialog box could be introduced, alerting the user to unique circumstances each time an unsigned executable is started with administrative privileges for the first time. This would alert the user to the unusual situation, particularly if an application which has been run as an administrator consistently in the past (such as games that require updates) suddenly display the new dialog.
- The Start Menu could display both Start Menu entries, which would likely be seen as unusual by the user.
- Disable reads from UAC-protected areas by applications that do not exist in the same protected path (so that applications that do not run from c:\Program Files could not read within c:\Program Files by default).
- Disable .NET Framework compilation in non-elevated processes by default, although this would not protect from similar attacks made from other languages, such as C, and could present difficulty for some applications which utilize dynamic compilation for extensibility.

As always, users will need to be vigilant and cognizant of the software they are running, particularly that which is downloaded from the internet. Unfortunately, as much as Windows will try to protect the user from malicious software, erring on the side of user freedom will always present a hole in user security. Still, Microsoft can do its best to implement the heuristics necessary to determine if such an attack is in

place and warn the user appropriately. Microsoft's existing policy on this type of issue, which they label the "10 Immutable Laws of Security³," would indicate that the exploit as outlined in this document is not appropriate for reporting; the first law states that running an untrusted program is tantamount to handing one's computer to a hacker.

However, if such a policy were truly accurate, Microsoft would have had no need to implement UAC in the first place; what UAC really does is minimize the total impact on a machine in the event that it is compromised. When a malicious program is executed as a standard user, any of its actions that could result in truly problematic consequences (such as writing to the Windows folder or the machine-wide registry) are virtualized. Consequently, work for support personnel is minimized. If untrusted programs were never run, minimizing their impact would be a secondary issue. This attitude is problematic, and with the hardening Windows Vista has introduced as part of its security push, Microsoft needs to be aware that exploits such as this will likely become more prevalent in at least the near future.

About the Proof of Concept Code

The Proof of Concept code attached is a Microsoft Visual Studio 2005-based solution. There are several projects that illustrate distinctive parts of the exploit. The Proof of Concept has been tested on Windows Vista Ultimate Edition for both x86 and x64-based systems (using WOW64⁴) and has found that the exploit is equally effective on both platforms.

The ExploitLib project (Visual C#)

ExploitLib is an *extremely* simple class library that contains a class that is activated and used once the stub executable is run with administrative privileges (step 8 of the generalized workflow in Figure 6, above). The ExploitLib project simply writes a short text file to the UAC-protected Windows System folder (by default c:\Windows\system32) called "exploited.txt". It is worthwhile to note, however, that this library could readily be any class library that contains any number of types that define a static method called "Execute".

The POCApp project (Visual C#)

The POCApp project is a skeleton of the Proxy Infection Vector application and the stub executable generator. It contains all of the logic necessary to traverse the relevant Start Menu folders and create corresponding hierarchies in the user Start Menu folders, as well as code for dynamically creating the stub executable. It contains the ExploitLib file as an embedded resource, deploys the stub executables, and does the necessary work to verify that a target executable is eligible for replacement (that it is not signed and is not a Windows OS-supplied file; it does not check system policy for untrusted executable signatures).

It is worthwhile to note that the POCApp project is probably not production-quality code. There are some checks it does not perform, such as whether it is overwriting a stub assembly because of a naming

³ <https://www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.aspx?mfr=true>

⁴ WOW64 is short for Win32 on Windows x64, which is the x86 compatibility layer in Windows Vista x64.

collision. However, for someone determined to do a level of damage, this would not be a particularly difficult additional task to implement.

It also does not scan all of the potential folders which can contain replaceable files. Potential other targets for this attack would include the Desktop, the Quick Launch bar, the Documents and other related folders (particularly Downloads), and other user-specific areas. As another exercise, the application could potentially move executables from their normal locations (on the Desktop or in the Downloads folder, for instance), replacing them with shortcuts, which would still potentially be transparent to most users.

About the Researcher

Robert Paveza is a senior web application developer with Terralever, a web-based marketing corporation located in Tempe, AZ. His blog can be found at <http://geekswithblogs.net/robp/>.