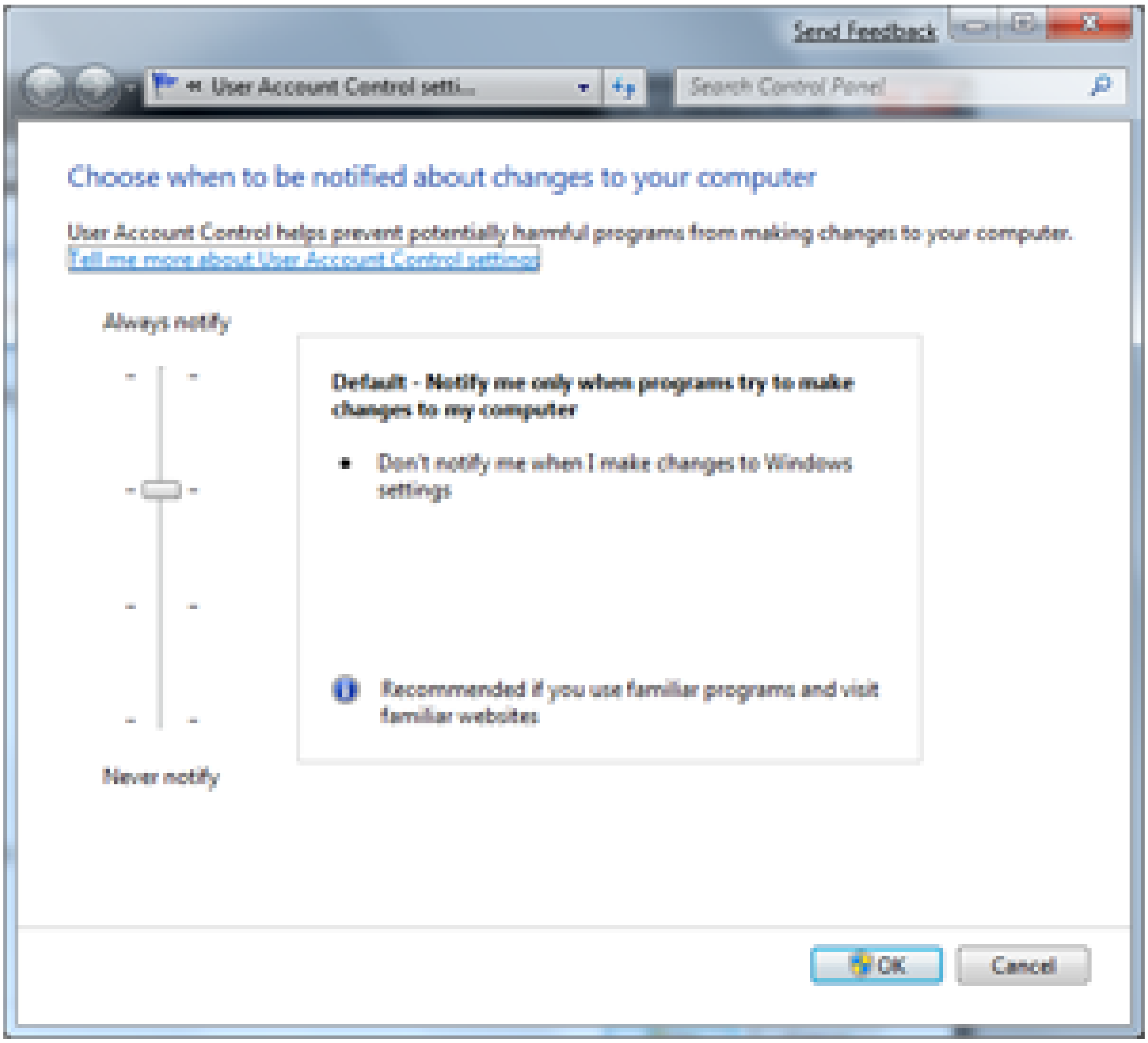
SUBSCRIBE SIGN IN

BIZ & IT —

Windows 7 UAC flaws and how to fix them

A number of security flaws have been found in Windows 7's streamlined UAC— ...

ARS STAFF - 2/8/2009, 7:00 PM



Choose when to be notified about changes to your computer

User Account Control helps prevent potentially harmful programs from making changes to your computer.
Tell me more about User Account Control settings

Always notify

Default - Notify me only when programs try to make changes to my computer

- Don't notify me when I make changes to Windows settings

Recommended if you use familiar programs and visit familiar websites

Never notify

The Windows 7 UAC Slider

Unlike many, I'm a big fan of Vista's User Account Control. Truth is, I don't get a lot of prompts asking me to elevate, and those that I do get are legitimate. Sure, the implementation isn't perfect; there are some scenarios that cause a rapid proliferation of prompts that are a little annoying (such as creating a folder in a protected location in Vista RTM), and there are even a few places where it forces elevation unnecessarily, but on the whole I think it's a good feature.

The basic purpose of UAC is to annoy you when your software needs Admin privileges. The reason for this is simple: a lot of Windows software demands Admin privileges not because it needs to be privileged for everything it does, but rather because it was the quickest, easiest way for the developer to do some minor task. For example, games with the PunkBuster anti-cheat system used to demand Administrator privileges so that PunkBuster could update itself and monitor certain system activity. This was bad design because it meant that the game was then running with Administrator privileges the whole time—so if an exploit for the game's network code was developed, for example, that exploit would be able to do whatever it liked.

The solution to this kind of problem is to split the application up in one way or another. In the PunkBuster case, the privileged parts were split into a Windows service (which has elevated privileges all the time), leaving the game itself running as a regular user. There are a number of other approaches of tackling the same problem, but in general they all require an application to be restructured somewhat so that privileged operations can be separated from non-privileged ones.

As well as this "annoyance" role, UAC also provides a warning when software unexpectedly tries to elevate its privileges. UAC has heuristics to detect applications that "look like" installers, and it also traps important system utilities like Registry Editor. Though Microsoft has cited this kind of behavior as a benefit of UAC, the company has also said that UAC is not a "security boundary." That is to say, if a malicious program figures out a way of elevating without a UAC prompt (or by tricking the user into agreeing to the UAC prompt) then that's *not a security vulnerability*. If you want real security with UAC you have to run as a regular user and enter a username and password to elevate—the Admin approval click-through mode (the mode that's the default for the first user account created on any Vista system) is not intended to be secure.

The winds of change are blowing

Why bring this up? Well, first of all, Windows 7 brings some changes to UAC to try to reduce the number of prompts that Administrators see. The basic idea is that if you make a change through one of the normal Windows configuration mechanisms—Control Panel, Registry Editor, MMC—then you don't get a prompt asking you to elevate. Instead, the system silently and automatically elevates for you. Third party software will still trigger a prompt (to provide the warning/notification that it's raising its privileges), but the built-in tools won't. In this way, you don't get asked to confirm *deliberate* modifications to the system; the prompts are only for unexpected changes.

In my na�vet� I initially assumed that perhaps the differentiation was made according to where the action initiated; keyboard and mouse input (i.e., user actions) rather than something more simplistic like trusting particular applications. After all, the computer knows that a keystroke or mouse click originated in the hardware (because a driver gets to handle it), so it can easily tell what's real and what's not. A trusted application, however, could be started up by a malicious program and made to do bad things. So surely that wasn't the route Redmond chose?

It turns out that is indeed the route Redmond chose. For a number of years now, Microsoft has attached digital signatures to the programs and libraries that make up Windows; these signatures allow you to verify that a program did indeed come from Microsoft just by looking at the program's properties. In Windows 7, most programs with Microsoft signatures are trusted by UAC and won't cause a prompt. Instead, they just silently elevate. Unfortunately, Microsoft hasn't done anything to resolve the problem with this approach—trusted applications can be tricked into doing bad things. A few programs such as cmd.exe, PowerShell, and Windows Scripting Host don't auto-elevate (because they're designed to run user code, rather than operating system code), but they're the exception. Everything else elevates, and is vulnerable to being abused.

This was noticed a last week by Long Zheng at I Started Something. Together with Rafael Rivera, he put together an exploit for this silent elevation. The exploit programmatically passed keystrokes to an Explorer window, navigating to the UAC Control Panel, and setting the slider to disabled. Because Explorer is trusted, changing the setting doesn't cause a prompt. Instead, UAC is silently disabled.

Sending keystrokes is a bit crude, so a second attack was developed. This second attack was more flexible; instead of merely disabling UAC, it allowed any code to run elevated without prompting the user. It does this by using a Windows program called rundll32. rundll32 has been part of Windows for a long time; its purpose is, as the name might imply, to allow DLLs to be run, almost as if they were normal programs. The exploit simply puts the malicious code into a DLL and tells rundll32 to run it. rundll32 is trusted, so it elevates automatically.

Together, these attacks mean that Windows 7's default UAC configuration is virtually worthless. Silently bypassing the prompts and acquiring Administrator privileges is as easy as putting code into a DLL. Windows Vista doesn't have a problem, because it doesn't trust any programs; the problems are purely due to the changes Microsoft has made to UAC in the name of convenience in Windows 7.

Dismissing instead of fixing

> Given the importance of security and UAC, one might expect Microsoft to take note of this problem and do something to fix it. Unfortunately, the company's first response was to dismiss the behavior as happening "by design."

Given the importance of security and UAC, one might expect Microsoft to take note of this problem and do something to fix it. Unfortunately, the company's first response was to dismiss the behavior as happening "by design." Redmond says that, because UAC isn't a security boundary, it doesn't matter if silent elevation occurs; it's not a vulnerability. UAC is only there to keep legitimate software authors honest, not to stop malware. After the second exploit was disclosed, on Thursday a company representative made a lengthy blog post reiterating that UAC is not a security boundary and that the behavior is by design—it's awfully *convenient*, you see, so it doesn't matter if it's actually *useful* as a security measure.

In essence, the argument Microsoft has made is that if a user runs malicious programs as an Administrator and those programs do malicious things, that's not a security flaw, because the user ran the programs as an Administrator, and an Administrator is allowed—by design—to do things that can break the system. What this argument misses is that, until elevated, the malicious program *can't* do all the nasty things that malicious programs tend to do; it can't modify system files, make itself run on startup, disable anti-virus, or anything like that. Choosing to run a program without elevation is not consent to running it elevated.

Maybe this needs to be fixed after all

Things then took a turn for the weird. A second post was made admitting that, well, the company had "messed up" with the first post, in two ways. First and foremost, the new UAC behavior is badly designed; second, the whole issue was badly communicated by the company. The Windows 7 team will change the UAC behavior from that currently seen in the beta to address the first flaw. This fix won't be released for the current beta, though, and we'll have to wait until the Release Candidate or even RTM before we can see it in action.

When fixed, the UAC control panel will be different in two important ways. It will be a high integrity process—which will prevent normal processes from sending simulated keystrokes to it—and changes to the UAC setting will all require a UAC confirmation, even if the current setting does not otherwise require it. Though this will resolve the first exploit, it looks like it will have no impact on the second, and since the second exploit was the more useful anyway (as it can be used to do anything, not just change the UAC setting), this fix doesn't seem extensive enough.

> There is some irony in Microsoft's behavior to use a trusted executable model; the company knows damn well that trusted executables aren't safe, and uses this very argument to justify the UAC behavior in Vista.

In short, trusting executables is a poor policy, because so many executables can be encouraged to run arbitrary code. There is some irony in Microsoft's behavior to use a trusted executable model; the company knows damn well that trusted executables aren't safe, and uses this very argument to justify the UAC behavior in Vista. A system using trusted executables will only be secure if all of those executables are unable to run arbitrary code (either deliberately or through exploitation).

That clearly isn't the case in Windows 7; rundll32's express purpose is to run arbitrary code! Removing the auto-elevation from rundll32 may be unpalatable, too. While non-elevating programs like Windows Scripting Host and PowerShell are used predominantly for user code, rundll32 is used mainly for operating system code. Removing its ability to elevate would, therefore, reintroduce some of the prompts that Windows 7 is trying to avoid. And even if rundll32 lost its ability to elevate automatically, there are almost certain to be other trusted programs that can be abused in a similar way. So, in spite of the most recent blog post, this remains a poorly-designed feature. UAC is now only as strong as the weakest auto-elevating program.

It equally remains poorly communicated. Fundamentally, the defense that UAC is not a security boundary just doesn't cut the mustard. Microsoft sells UAC as providing "heightened security", as a way of limiting the "potential damage" that malware can do. To then argue that users should not, in fact, expect UAC to keep them secure is insulting. Moreover, even if the purpose of UAC is just to keep application writers honest, these exploits mean it fails to achieve even that. The simple fact is that it's a lot easier to restructure an application to make it use rundll32 to automatically elevate than it is to do things the Right Way. The unscrupulous or lazy software vendor who just wants to do the simplest thing possible to make the prompts go away will surely prefer that option to actually fixing their application.

As someone who thinks that UAC is a good idea, these efforts to undermine it are terribly disappointing. As things currently stand, Windows 7's default UAC settings render it pointless in Admin approval mode, as it's so trivially bypassed. It might as well be turned off completely for all the good it does. To break a security feature—boundary or no boundary, it's sold as a security feature, it acts like a security feature, so I'm certainly going to treat it as a security feature—for the sake of convenience is a grave mistake.

READER COMMENTS 117

SHARE THIS STORY

Advertisement

WATCH

Modern Vintage Gamer Reacts To His Top 100...

Modern Vintage Gamer Reacts To His Top 1000 Comments On YouTube

We searched through Modern Vintage Gamer's most popular videos, picking the top 1000 comments based on number of likes, first comments, and frequently asked questions. Then, we got the man himself to sit down and take us down memory lane. Ranging from the earliest days of his channel right up to the present, we've woven together MVG's personal history on YouTube - and captured his reactions to the whole thing.

Modern Vintage Gamer Reacts To His Top 1000 Comments On YouTube

How The NES Conquered A Skeptical America In 1985

Scott Manley Reacts To His Top 1000 YouTube Comments

How Horror Works in Amnesia: Rebirth, Soma and

More videos

← PREVIOUS STORY NEXT STORY →

Related Stories

Sponsored Stories

Powered by

Today on Ars

STORE
SUBSCRIBE
ABOUT US
RSS FEEDS
VIEW MOBILE SITE

CONTACT US
STAFF
ADVERTISE WITH US
REPRINTS

NEWSLETTER SIGNUP

Join the Ars Orbital Transmission mailing list to get weekly updates delivered to your inbox.

SIGN ME UP →