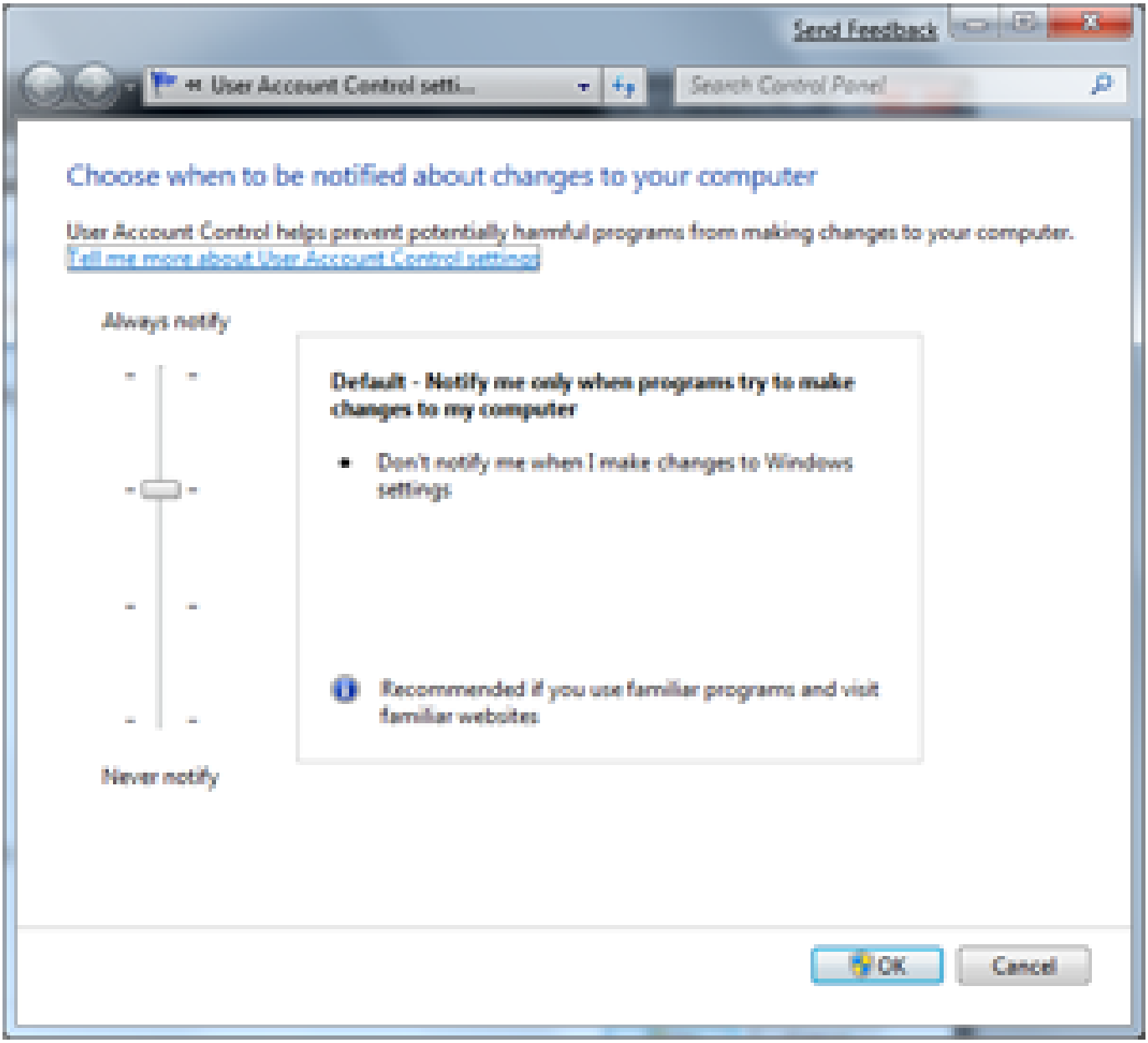
BIZ & IT —

Opinion: Windows 7's UAC is a broken mess; mend it or end it

The changes Microsoft has made to Windows 7's UAC render it little more than a ...

ARS STAFF - 3/5/2009, 12:31 AM



The Windows 7 UAC Slider

I wrote a few weeks ago about changes Microsoft has made to Windows 7's User Account Control (UAC) that make the component less secure than it was in Vista. Though the company has responded by saying it will change some of the problem behaviors, yet more problems have emerged that indicate that a real fix will be harder than first expected. But more than that, the flaws call into question the entire purpose of the Windows UAC feature, at least in its commonplace "Admin Approval" mode.

The decisions Microsoft has made not only make Windows 7's Admin Approval mode less secure than Vista's, they also undermine the entire purpose of the UAC system. Redmond maintains that UAC's foremost objective is to ensure programmers update their programs to behave properly when users have limited access rights. But the way that the Windows 7 UAC "improvements" have been made

completely exempts Microsoft's developers from having to do that work themselves. With Windows 7, it's one rule for Redmond, another one for everyone else.

The combination of significant security flaws and the inconsistent, "Do as I say, not as I do" attitude towards UAC should give Microsoft pause for thought. There's no point in retaining Admin Approval mode as it currently stands, and it should be scrapped completely.

The new exploit, discovered and demonstrated here, depends on a third mechanism for elevation that was previously overlooked. The first mechanism for elevation is the traditional prompt—the user is notified that a particular program wants to elevate, and can permit or deny the request. The second is the auto-elevating executables described in my previous article, in which certain system executables automatically elevate without any notification. Chief among these is a program rundll32, which can load and run almost any DLL, and will do so fully elevated.

Microsoft may or may not fix the rundll32 problem; as it stands, it blows a big hole in UAC since it allows any software to trivially bypass the prompts, but since the change was made with the objective of removing prompts from "legitimate" uses of rundll32, the company has something of a dilemma: stop rundll32 auto-elevating and reinstate the prompts (thereby improving security), or keep the auto-elevation and ignore the security impact.

It may not matter much what Microsoft does with rundll32, however, as the newly demonstrated attack shows. The new attack allows an attacker to trick pretty much *any* auto-elevating program into running code of the attacker's choosing—even auto-elevating programs that aren't meant to run arbitrary code. It does this by exploiting other parts of Microsoft's auto-elevation system.

Overview of the new attack

Although a few programs in Windows 7 are always elevated, most are not. For example, the Explorer shell runs without elevation, unless the user explicitly opts to elevate it and verifies the UAC prompt. Nonetheless, there are Explorer tasks that require elevation that are common enough that Microsoft felt they should auto-elevate. The most common one of these is probably creating a folder in a protected location (in Program Files, for example). In the original Vista release, this activity would

cause an annoying back-to-back double elevation: once to create the folder, and again to rename it to its intended name. Service Pack 1 streamlined this a little, reducing it to only a single elevation, but Microsoft clearly wanted to get this down to zero.

The technique that all versions of Vista and Windows 7 use to perform individual tasks with elevation (rather than running an entire program elevated) is to put the elevated action into its own component and to call that component from the main program. This is in fact the main way in which UAC support should be added to applications, because it generally requires less elevation than elevating entire programs. If the operation in question isn't even attempted, no attempt to elevate occurs either, which is obviously the best possible outcome.

This component-based technique is used for Explorer's file management operations in Vista and Windows 7. Creating, copying, moving, renaming, and deleting files all occur within a particular component that gets elevated when necessary, leaving Explorer itself unelevated. In Windows 7, however, Microsoft has made this component auto-elevating. So although Explorer itself cannot elevate automatically, it can create a component which can.

This component is quite limited—it can do a handful of file manipulation operations, but won't run arbitrary code—and even the auto-elevation is restricted. Auto-elevating components will only elevate when called from Microsoft-signed applications; if third-party code tries to use them, a UAC prompt will appear. On the face of it, this wouldn't be enough to compromise a system; third party code can't use the component to elevate, and even if the component is running, it can't be used to trivially run arbitrary code in the way that the rundll32 flaw can (although it could certainly overwrite or remove key system files, which might break the system).

Unfortunately, the "Microsoft-signed application" restriction is easily bypassed using a standard Windows trick that allows one process to insert code into a second process, as long as both processes are being run by the same user. The limitations of the file management component are probably unavoidable (it can only do the things it has been programmed to do, after all), but it turns out it doesn't really matter. The file management component can place files into various locations on the system that an unelevated user cannot; an auto-elevate program can then be tricked into loading those files and executing code from them.

The result is, just as with the rundll32 problem, silent and automatic elevation, able to do anything.

The implications

So, does any of this matter? Well, I think it does. Microsoft and its supporters have argued throughout that UAC in Admin Approval mode isn't a security boundary, and as such, escalation of this kind is not a security problem. Although Windows does have plenty of security boundaries—two users logged on at the same time should not be able to kill each other's processes or read each other's data, for example, because each session has a boundary around it—UAC is not one of them. What this means is that it doesn't really matter, in Microsoft's view, if people figure out a way to bypass UAC.

And indeed, in Vista there are ways for malicious programs to piggy-back off UAC elevations to get elevated themselves, and these haven't been fixed. There is, however, a big difference between how this plays out in Vista vs. Windows 7. In Vista, these workarounds still depend on the user at some point permitting a program to elevate, and the elevated program has to be the one that the malware has booby-trapped. In Windows 7, all the guesswork is gone; the exploitation is consistent and systematic.

Microsoft hasn't been entirely consistent in its stance on this matter. The company has bowed to public pressure over some of the Windows 7 UAC changes already, and reinstated more secure behavior even though this has meant reintroducing some UAC prompts. This move is inconsistent with the stated policy; after all, if UAC is truly not a security barrier, why bother making fixes whose only justification is the security they provide? However, the latest exploits appear to be essentially unfixable without wholesale reintroduction of the UAC prompts. Since the entire motivation behind the changes in the first place was to avoid these prompts, any solution that reinstates them is unlikely to fly.

Page: 1 2 Next →

READER COMMENTS 107 SHARE THIS STORY

Today on Ars

STORE
SUBSCRIBE
ABOUT US
RSS FEEDS
VIEW MOBILE SITE

CONTACT US
STAFF
ADVERTISE WITH US
REPRINTS