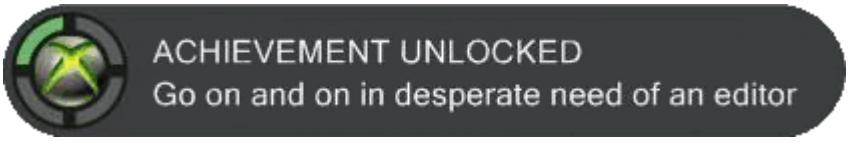
Windows 7 UAC whitelist: Code-injection Issue (and more)

Archived: 2026-04-05 20:17:34 UTC

Windows 7 UAC whitelist: **- Code-injection Issue** **- Anti-Competitive API** **- Security Theatre**	

(Still needs an edit/re-write/chainsaw but I can't be bothered. I have wasted enough time on this already.)

As I was typing more words into this page, this appeared in my text editor at the 10,000th word! :-)



Quick Windows 7 RTM update:

Everything below still applies to the final retail release of Windows 7 (and all updates as of 14/Sep/2011).

Quick Windows 8 update:

Everything below still applies to the Windows 8 Developer Preview released on 13/Sep/2011. It is early days, of course, but from a quick look it does not seem that anything UAC-related has changed at all in Win8.

Contents:

- UAC Comparison: Two file-managers
- If a whitelist makes sense then it must be user-configurable
- Previous Windows 7 UAC issues
- To those saying, "but it requires code to get on the box"
- To those saying, "but UAC isn't a security boundary"
- To those saying, "but it's only a beta"
- Quick response to a couple of newer things

Program, Source Code and Step-by-Step Guide

While Windows 7 was still in beta Microsoft said this was a non-issue, and ignored my offers to give them full details for several months. so there can't be an issue with making everything public now.

March 2020: Binaries removed as I got sick of the page being marked as malware, even by Google (FFS) who themselves publish thousands of similar disclosures. There's no way to contact a human being at Google. The old binaries still worked completely on Windows 10 in the year 2020 and the issue is never going to be addressed by Microsoft, but apparently providing proof of it is bad in some way. Whatever. The source code remains up but you can find the binaries yourselves. If you end up getting ACTUAL malware because you couldn't get the binaries from a safe, official source, blame Google.

- Win7ElevateV2.zip (32-bit and 64-bit binaries; use the version for your OS.
- Win7ElevateV2_Source.zip (C++ source code, and detailed guide to how it works.)
- Source in HTML format (for browsing online)
- Step-by-step guide (description of what the code does)

This works against the RTM (retail) and RC1 versions of Windows 7. It probably won't work with the old beta build 7000 due to changes in which apps can auto-elevate.

Microsoft could block the binaries via Windows Defender (update: they now do via MSE), or plug the CRYPTBASE.DLL hole, but unless they fix the underlying code-injection / COM-elevation problem the file copy stuff will still work. Fixing only the CRYPTBASE.DLL part, or blocking a particular EXE or DLL, just means someone has to find a slightly different way to take advantage of the file copy part. Finding the CRYPTBASE.DLL method took about 10 minutes so I'd be surprised if finding an alternative took long.

Even if the hole is fixed, UAC in Windows 7 will remain unfair on third-party code and inflexible for users who wish to use third-party admin tools.

Microsoft Security Essentials detects proof-of-concept as "hacktool"

As originally reported by Maurice at AeroXperience, the free Microsoft Security Essentials anti-virus tool now detects the Win7ElevateV2 binaries as **HackTool:Win32/Welevate.A** and **HackTool:Win64/Welevate.A**.

Apparently recompiling the binaries in VS2010 means they are no longer detected. So as well as being named after the proof-of-conept executables the detection seems to be looking for those executables specifically, rather than detecting a general pattern within them. That seems pointless when, aside from demonstrating what a program *could* do, the proof-of-concept executables don't do or allow the user to do anything that couldn't be done

using Windows Explorer. The point of them is to demonstrate that anything Windows Explorer is allowed to do can be hijacked by any other program.

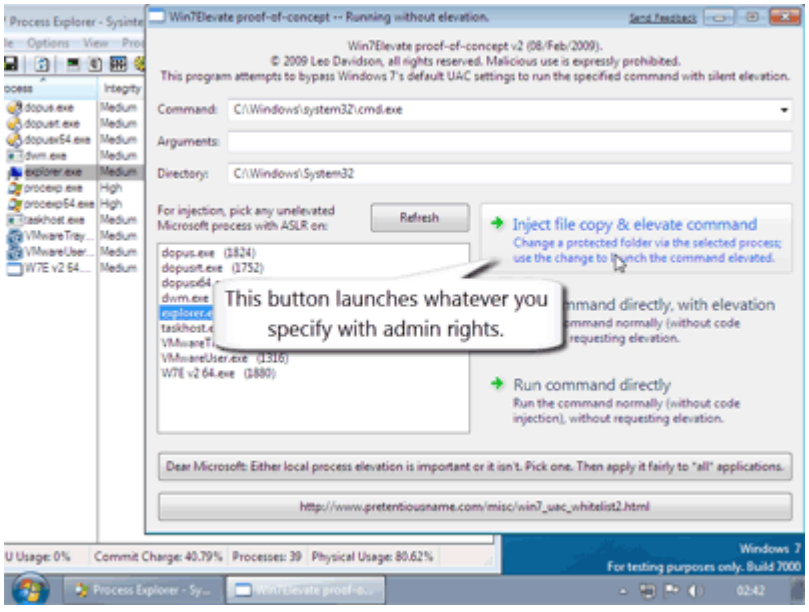
Another program using the same technique would not be detected by MSE's current anti-virus signatures.

If this whole thing is a non-issue then why block the proof-of-concept binaries? On the other hand, if bypassing UAC does makes something dangerous then shouldn't Explorer.exe be detected and blocked as well? :-)

Win 7 UAC Code-Injection: Video demonstrations

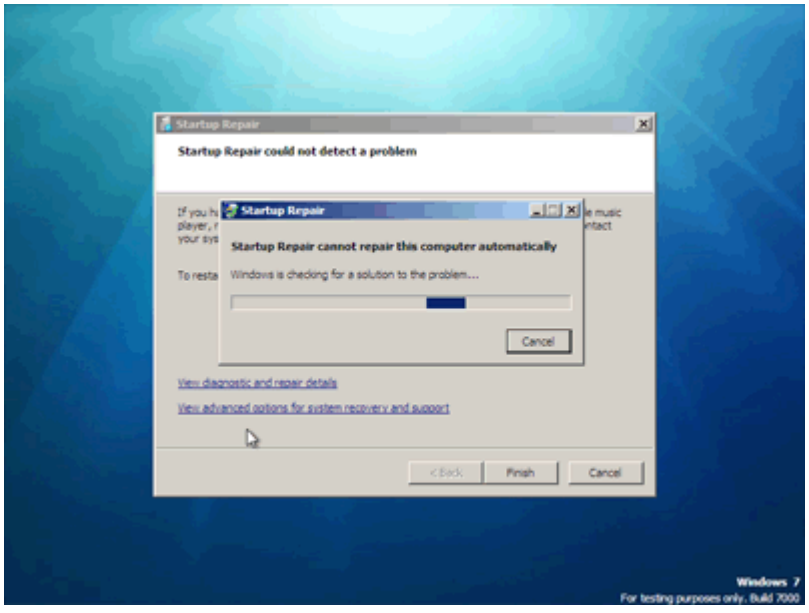
I'll cut to the chase for the TLDR crowd.

- **Video demonstration of Win 7 beta UAC flaws and design:**



Mirror 1; Mirror 2

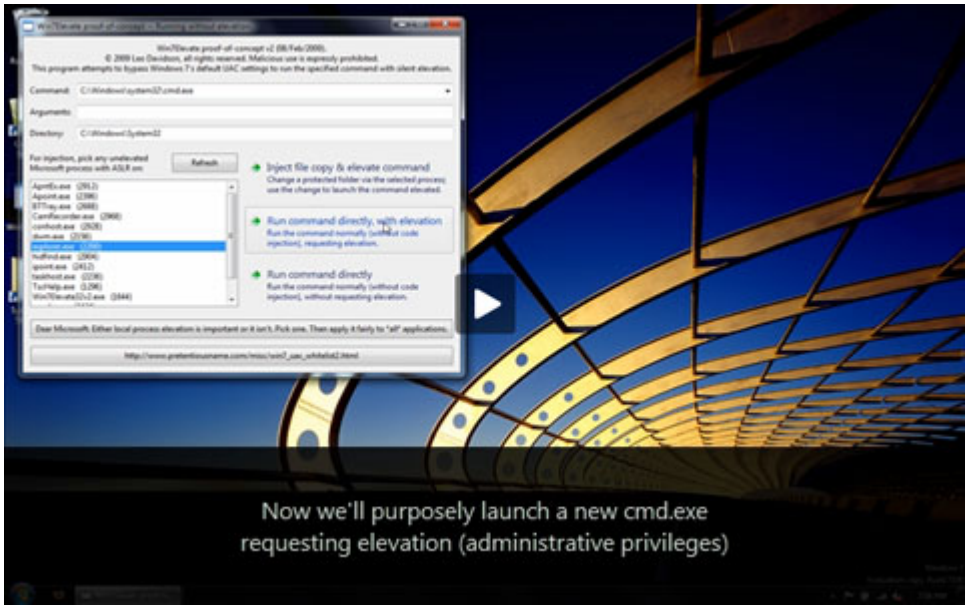
- **A more dramatic example of a machine being hosed:**



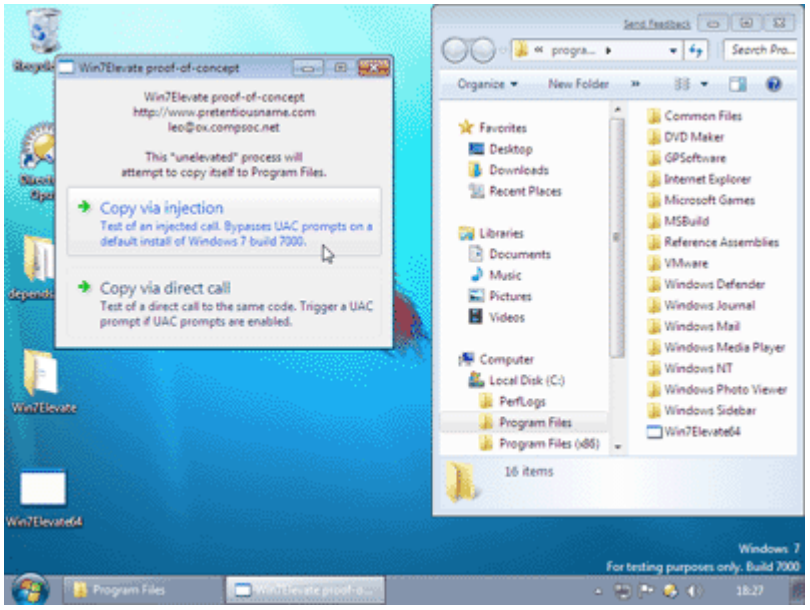
Mirror 1; Mirror 2

Depending on which side of the "trust all code on the box" fence you are on, watching the two videos above should show you *either* how serious this problem is *or* how pointless it is to force UAC prompts on third-party apps.

- **Long Zheng's video demonstration using Win 7 RC1 build 7100:**



- **The original, less interesting version which just copied a file:**



Mirror 1; Mirror 2

(Thanks to Kelbv and Jon for hosting the videos.)

Some quotes

(Bold emphasis is mine.)

User Account Control (UAC) is a core security feature *in the next release of Windows Vista and Windows Server code name Longhorn.*

--Microsoft's UAC blog

*The other fact is that **User Account Control (UAC) is one of the most important ways that we hope to protect people in Windows Vista**. [...] Once the OS is released, if you absolutely can't stand **a security feature that is designed to protect you**, by all means, turn it off*
--Jesper's Blog (Senior Security Strategist in the Security Technology Unit at Microsoft.)

Here's my million dollar question: If UAC wasn't designed to ultimately protect us from anything, why does its icon resemble a damn shield?
--Rafael's Within Windows

Standard user accounts are a distraction and an excuse as far as Windows 7 UAC goes. You might as well say "People should use Linux to be more secure" as it's about as relevant and likely to happen. If Windows 8 (or whatever) actually makes standard user the default, and improves the user experience to one that people might actually put up with, then the argument will hold water.
--Me :-)

Win 7 UAC Code-Injection: Summary

On 5th February 2009 I wrote a proof-of-concept program to demonstrate a flaw in Windows 7's UAC under default UAC/account settings. This simply copied a file to Program Files without the user's consent. In other words, it performed a file copy to a protected location from an unelevated process, bypassing UAC.

"So what? All it does is copy a file?"

On 9th February 2009 I extended the proof-of-concept program into to something which, without requiring elevation itself, can run and elevate any command or program without triggering a UAC prompt.

All of this is done *without* using the SendKeys or RunDll32 holes which were found earlier in February. It is done using a method which can attack almost any Windows executable and which is inherent to the changes Microsoft have made to UAC in Windows 7.

The proof-of-concept works on unmodified installs of Windows 7, including the October retail release, both 32-bit and 64-bit versions, at default settings.

Setting UAC to its highest level, or using a non-admin account, will prevent the proof-of-concept from working by forcing it to display a UAC prompt. However, neither of those are the default settings of a Windows 7 install.

As well as discussing the proof-of-concept code I argue that:

- Microsoft should *either* admit that local process elevation is a problem and make Windows more secure by default *or* admit that the Windows 7 default UAC settings are security theater (as they offer no protection) and anti-competitive (as they are inflicted on third-party code despite local elevation supposedly being a non-issue).
- If there is to be a UAC whitelist, or the equivalent of one, then it should be up to the user which Microsoft and third-party software is on it. Users should not be forced to expose themselves to risks from software they do not use. Conversely, if reducing UAC prompts in frequently-used software is needed to stop people

disabling UAC entirely then that applies to third-party software as much as to bundled software (especially once a machine is past the "setup" phase).

- The "elevate without prompting" UAC option should be available in the main UAC control panel, in particular so that Home Premium users can access it. (This allows you to turn off UAC prompts without turning off the undeniably good parts of UAC such as Protected-Mode IE.)
- UAC itself *was* a good API and a good design that was given a bad name because of the way it was used by Microsoft's application-level code (such as Explorer and Control Panel). Accordingly, the user experience of having UAC enabled could have been vastly improved by changing the application-level code without opening a huge hole in UAC.
- Microsoft created these problems themselves and, rather than fixing them properly, have taken the easy way out, unnecessarily making UAC less secure in the process. At the same time Microsoft expect third-party vendors to do a better job than they bothered to do using the API which they themselves designed.
- If UAC is inherently "undefendable" then that applies equally to UAC when used for elevation from a standard user account.
- If the only goal of UAC is to coerce third-party programmers into working better under standard user accounts then there are far, far better ways to achieve that.
- Microsoft need to get their message straight about what UAC is and isn't supposed to do. They need to stop saying that UAC is a security feature only to say it isn't, depending on what suits them at the time.

And, for the record, I like Windows and much of what Microsoft do, in general. I even like UAC (the API, *not* the way it has been used). I wrote this page because I care about the platform not because I get a kick out of attacking something Microsoft have done. I call things as I see them. I attack and criticise some of what Microsoft do and I support and defend Microsoft other things that they do.

Win 7 UAC Code-Injection: The good news

All of this only affects the default account type and UAC level of Windows 7 (builds 7000 & 7022, but probably also the retail given Microsoft's stance so far). If you go against the defaults and run as a non-admin user or turn UAC up to the *Always Prompt* level, so it behaves like it did in Vista, then it is no longer possible for code-injection from unelevated processes to bypass UAC prompts. So the advice remains as before:

> *If you are using Windows 7 and want to be protected against silent elevation then turn UAC up to the highest level.*

However, I would add to that advice:

> *If, on the other hand, you don't care about silent elevation then you should turn down UAC to Elevate Without Prompting -- so that UAC is still enabled but it never prompts you -- because the default level isn't buying you much except a few pointless prompts which can be bypassed by any program which wants to.*
>
> The *Elevate Without Prompting* mode, which was available on Vista as well, can be set via the Local Security Policy control panel (screenshot) but isn't one of the options in the simplified UAC control panel.

You either care about silent elevation or you don't. Pick one.

Win 7 UAC Code-Injection: How it works

In the quest to reduce the number of UAC prompts, for their code only, Microsoft have granted (at least) three groups of components special privileges:

1. **Processes which anything else can run elevated without a UAC prompt.**

   This is the list of about 70 processes published on Rafael's Within Windows blog. (Update: New list for RC1 build 7100.) If you run a process on this list and it requires elevation then it -- the whole process -- will be given elevation without showing you a UAC prompt.

   Discovery of this list is what lead to the earlier RunDll32.exe exploit where you could ask RunDll32.exe to run your code from within a DLL and it would do so with full elevation and no UAC prompt. Microsoft have since removed RunDll32.exe from the list but there are still plenty of other processes on the list, several of which can be exploited if you can copy files to the Windows folder.

2. **Processes which can create certain elevated COM objects without a UAC prompt.**

   Programs on this second list are able, without being elevated themselves, to create certain elevated COM objects without triggering a UAC prompt. Once such an object has been created the processes can then tell it to perform actions which require administrator rights, such as copying files to System32 or Program Files.

   This appears to be a superset of the first list. In fact, it seems to include all executables which come with Windows 7 and have a Microsoft authenticode certificate.

   Unbelievably, as of build 7000 (and confirmed in RC1 build 7100), the list includes not only programs like Explorer.exe which use this feature (or potential security hole, if you like) but also programs such as Calc.exe, Notepad.exe and MSPaint.exe. Microsoft appear to have done nothing to minimize the attack surface and have arbitrarily granted almost all of their executables with this special privilege whether they actually use it or not. You can see evidence of this yourself by opening MSPaint, using the File Open dialog as a mini-file manager, and making changes within Program Files (e.g. create a folder or rename something); it'll let you do that without the UAC prompt that non-MS apps should trigger. I doubt that is intentional and it shows how little thought has gone into the UAC whitelist hacks MS have added to make their own apps seem better.

3. **COM objects which can be created with elevation, by the things in list 2, without a UAC prompt.**

   Full enumeration of this list has not yet been done. The list is known to include IFileOperation and may simply be all Microsoft-signed COM objects that allow elevation at all.

   It does not look like third-party COM objects can be elevated without triggering a UAC prompt, even by Microsoft processes, so the process and object must be on lists 2 and 3 respectively to bypass the UAC prompt. Given the number of processes which can be attacked and the fact that there are Microsoft COM objects to do many admin tasks, that isn't much of a consolation.

My proof-of-concept program is a standalone executable that is run as a normal unelevated process. I made from scratch in about a day and a half. Keep in mind that, while I am an experienced Windows developer, I am not a "security researcher" or "hacker" and this isn't the kind of thing I write every day.

The proof-of-concept works by directly copying (or *injecting*) part of its own code into the memory of another running processes and then telling that target process to run the code. This is done using standard, non-privileged APIs such as WriteProcessMemory and CreateRemoteThread.

If the target process is on list 2 (above) then our process gains the ability to create and control elevated COM objects from list 3 (above) without triggering a UAC prompt or giving any indication to the user (under default Windows 7 beta settings).

Typically, Explorer.exe is targeted because it is always running to provide the Windows taskbar and desktop. However, innocuous programs such as Calc.exe and Notepad.exe can also be targeted, launching them first if required. Like the proof-of-concept program, the targeted process is unelevated (and must be for direct code-injection to work).

The target process can have ASLR on or off and can be 32-bit or 64-bit; neither matters. (However, ASLR should mean that it would be difficult to launch a code-injection attack directly and reliably from a remote-code-execution exploit. Such an exploit would probably have to write-out and launch a separate EXE.)

The underlying problem is that the silent elevation feature, enabled by default in Windows 7 beta, does not check where the code requesting elevation comes from. It checks which process it is running within but not the particular code came from. So, for example, if you inject code into Explorer, or get Explorer to load your DLL, then you can create elevated COM objects without the user's knowledge or consent.

There are many ways to get your code into another process that's running at the same security level. That usually isn't a problem because there's usually nothing you can do in that other process that you can't already do in your own. The silent elevation feature changes that.

Getting your code into another process can be done in various ways. You can use buffer-overflow exploits (although ASLR helps greatly to mitigate those) or you can install yourself as a plugin DLL which the targeted program loads, like an Explorer shell extension. My proof-of-concept program uses a well-known technique called code injection. Code injection has the advantage that you don't have to trick the target program into loading your code; you simply push it into the other process and tell it to run it. This isn't a hack, either; everything is done using documented, supported APIs. (Legitimate uses of the APIs include debugging and inter-process communication. The APIs do not require elevation to use.)

(Note: The Wikipedia article on code injection is, at the time of writing, about a related but different type of code injection. Unlike what's described there, the code-injection technique I am using does not depend on feeding the target program a specially-crafted input that confuses it. Instead, I am calling Windows APIs which allow one process to copy code and data into another and then make that target process execute the code. The technique is often called *DLL injection* in the Windows world, but that name isn't accurate here because I am not injecting a DLL. I am copying, or *injecting*, the code directly from my in-memory exe to the target process. Update: A second

technique which does use a DLL is used as well now, but the main technique discussed here still doesn't invole DLLs. See the source archive for details.)

Update: Now that the Program and Source Code have been released you can download it or, read it online, to see exactly what it does. I have also written a separate step-by-step guide which goes into more detail about exactly what the source does, including the CRYPTBASE.DLL issue which had not been discussed before the source release (June 2009).

UAC in Vista and Windows 7: Mistakes then and now

Consider this combination:

1. **UAC in Vista was criticised mainly because of the excessive frequency of prompts.**

   Microsoft want to reduce the number of prompts. Nobody is against them doing that. The problem is *how* they are doing it.

2. **The excessive prompts were due to the way UAC was used at the application level, not the design of UAC itself.**

   In general, UAC *was* a well-designed, flexible and secure elevation API. However, converting old code to use UAC and provide a good user experience requires refactoring. Microsoft failed to do this in Vista and in particular with Windows Explorer, the Control Panels and Properties dialogs. Rather than refactor they retrofitted existing code and this resulted in excessive UAC prompts. (See "UAC Comparison" below.)

3. **UAC in Vista provided a good security layer between admin and non-admin code and user interfaces.**

   UAC was not a security boundary in the strict technical sense but it did do a good job of making it *very difficult* to cross from one side to the other without user consent. That is worth keeping.

4. **With Windows 7, Instead of modifying their application-level software to use UAC better, Microsoft have taken the easy way out and modified UAC to allow their software -- and only their software, or so they intended -- to elevate for free, without prompting.**

   In other words, Microsoft are *still* avoiding the refactoring work which their code has desperately needed since the early versions of Vista.

5. **In doing so Microsoft have created several obvious, easy-to-exploit and inherent flaws.**

   Had they created secure silently-elevating processes then, perhaps, they could have argued that their code deserved special trust; they could have argued that it was too dangerous to allow silent elevation for any third-party code. They failed *spectacularly* to so and thus cannot be allowed to argue either point. Beta or not, you design security in from day one, not at the last minute. Even if they patch over some of the security issues the fact is that the system is inherently flawed and more holes will be found in time. The best they can do, without drastic changes, is obfuscate things.

Another thing: The design of UAC in Vista shows that Microsoft had already predicted the very problems they are now opening up in Windows 7. Things which Vista's UAC went out of its way to prevent are now not only allowed but used routinely.

6. **The real aim of UAC, we are told, is to coerce developers into refactoring their software.**

   While trying to dismiss the holes they've opened up, some from Microsoft appear to be saying that UAC isn't supposed to be a security feature for admins at all. Instead, it seems, its aim is to coerce third-party developers into making their software to work better with limited user accounts and over-the-shoulder elevation, not requiring admin rights all the time (for that increases the attack surface), and elevating only when required without nagging the user too much...

7. **The eventual goal of UAC is to have everyone running as standard users, not admins, and yet:**
   - **Nothing has improved for non-admins.**
   - **The imbalance between admin and non-admin has grown.**
   - **If people complained about UAC as admin on Vista they'll complain even more about running as non-admin.**

   In the rush to respond to complaints about running as administrator it seems the situation for standard users -- the stated end-game for UAC -- was completely overlooked. Standard users still get just as many UAC prompts as before. Crucially, since non-admins (understandably) have to type an admin password for every prompt, running as standard user is more annoying than running as admin was on Vista. Since Microsoft know that running as admin on Vista was too annoying for many people they can't seriously think that people are going to switch to an even more annoying standard user account in Windows 7. If they wanted people to switch then they should have been working to improve the user experience of that mode (or all modes), not add silent elevation for admins and make it even more convenient for people to use the admin mode they want them to move away from.

   Leaving UAC as it was and refactoring the Windows applications & applets to use UAC better would have made Windows better for *all* types of user and would have encouraged people to move to non-admin accounts, all without creating such large security issues.

Now, wait a minute...

UAC exists to make developers change their code. Microsoft retrofitted UAC to their code but it annoyed too many people. Microsoft needed to refactor their own code but decided it was too hard (or failed to imagine how it could be done). So they left their code the same, and instead changed UAC to exempt themselves from having to do the job properly. Yet they still expect everyone else to do the hard work they failed to do, using the system they themselves designed. On top of this, in the quest to make their own apps less annoying with as little effort as possible, they have made the whole OS less secure under default settings and made the admin/non-admin choice even more weighted towards the admin option. And they have done nothing to improve the user experience of third-party apps under UAC for any type of user. Nor that of unbundled Microsoft apps such as the popular SysInternal tools which require elevation. Under default settings they are forcing UAC prompts on third-party code purely to maintain the *illusion* of security while giving their own poorly designed apps a free pass and dismissing new local code elevation issues.

Security has being reduced, the quest towards non-admin accounts has been harmed and third-party applications are being punished all for Microsoft's failure to properly design around their own API.

We should not accept this.

Microsoft: **Eat your own dog food before feeding it to us!**

(This section wouldn't be complete without a quick mention of the UAC Secure Desktop feature: Great idea but what went wrong with the implementation? It seems somewhat dependent on the videocard brand/driver and system load but, whoever is responsible, it simply isn't acceptable for the whole desktop to go unresponsive and turn completely black -- worse if you have multiple monitors -- for up to 10 seconds while Windows does who-knows-what before the dimmed desktop -- still annoying with large/multiple monitors -- and UAC prompt actually appear and allow you to continue (perhaps after another 10 seconds of black screen(s)). I really cannot believe that modern hardware has to take that long to switch from one picture and mouse/keyboard input queue to another. If that switch was implemented better then UAC would have been half as painful. That's one thing I do blame UAC itself for, or maybe a particular popular video card vendor if it's their fault, rather than the applications that use UAC.)

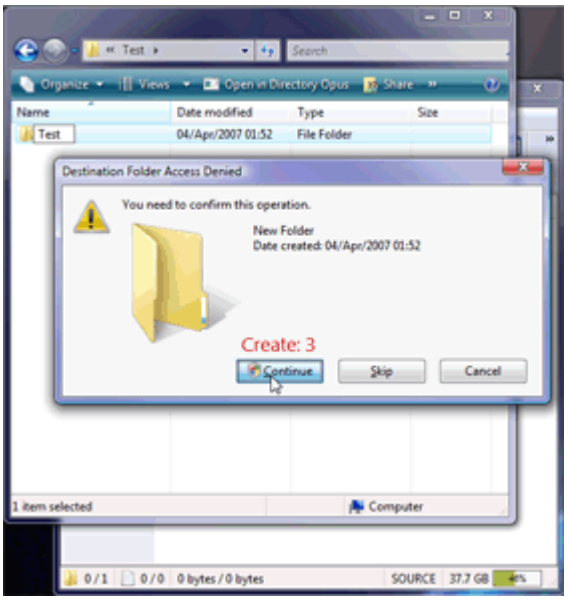
UAC Comparison: Two file-managers

I make the claim that UAC itself is okay and the problem has largely been how it has been used at the application level, particularly in Windows Explorer and the Control Panels. Since Vista's release UAC *has always* allowed applications to cache their elevated objects for as long as they need to. If the same program shows you two prompts it's because it was designed that way, not because UAC forced it to.

Here is some evidence to back that up.

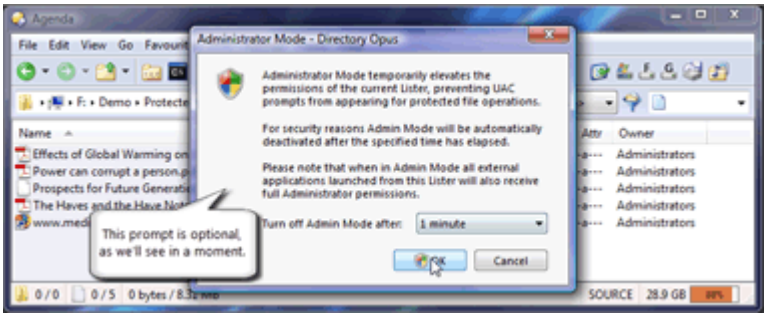
The two videos below were made back in 2007 just over two months after Vista's released. They compare two file managers on Vista: Windows Explorer and Directory Opus. Note how much better the user experience is when using Opus compared to Explorer. Then note how much longer Microsoft had to redesign their file manager for their API and wonder how they did such a bad job. Then wonder why they decided to improve nothing about the way UAC was used in Windows 7 and instead botched the default UAC mode to cover up their annoying prompts while still expecting everyone else to do a better job, now on an unlevel field.

(Vista SP1 was released since these videos were made and improved Explorer's handling of new folder creation. Perhaps someone saw my video? I don't know. So maybe this isn't a fair comparison today but note that only the new folder case was improved and the excessive prompting in Explorer continues to this day for other actions, even in Windows 7 (build 7000) if you disable silent elevation. Even if Vista SP1 had fixed the other cases, which it didn't, by then the bad taste had already been left in people's mouths.)



2007 video: See how Opus behaved compared to Explorer's gratuitous UAC prompting.

Opus, again back in 2007, also included an **Administrator Mode** option where you could click a button, consent to just one UAC prompt and then perform as many admin actions as you wanted to until a time limit (which you specified) ran out. A lot of people criticised UAC for not having a mode like this, like other platform's have, but there was nothing about UAC itself preventing such a thing from being implemented:



2007 video: Administrator Mode in Opus.
Note that Delete Confirmation was turned off for this video. The Delete Confirmation setting is independent of admin mode.

It's worth pointing out that Administrator Mode in Directory Opus does open as security hole. **If** you **choose** to use it and **consent** to the UAC prompt then you are sacrificing some security for some convenience while the mode is active. You will have a non-admin window from which you can trigger actions on an elevated COM object and that window is vulnerable to other non-admin applications sending it mouse and keyboard events. Also, if you run Opus buttons which you've configured to launch third-party apps (usually command-line driven batch-processing tools that are run on the selected files) then they will be run with admin access. The assumption is that if you switch into admin mode to run such a command then it's intended to modify the files and thus the command needs write access to them.

This is different to what Microsoft have done with their applications on Windows 7: On Windows 7 (build 7000 with default settings), Explorer (among other things) is **always** running in this insecure mode. There's no UAC prompt when you enter it (because you're always in it). You can't turn it on and of as and when you wish to perform a bulk of admin actions and sacrifice security for convenience, except by changing the system-wide UAC level (which is hardly convenient, either). In Opus you have to opt-in to use Administrator Mode by clicking a toolbar button (which isn't shown in the UI by default) and then consenting to the one-off UAC prompt. That UAC prompt in turn prevents other programs triggering the mode by themselves, though they could obviously wait for it to happen.

I'm not saying this Administrator Mode is right for all programs or even right for all Opus users. I rarely use it myself as I don't find the normal UAC prompts annoying, outside of exceptional circumstances. I'm just showing it as a demonstration that UAC is a flexible API and there are more ways to use it than many people realise.

If a whitelist makes sense then it must be user-configurable

(I talk about the concept of a whitelist here. As currently implemented, special elevation privileges are granted via manifest attributes and code signing, rather than a literal list, but conceptually the two are the same.)

Let's play a game of *what if...* I don't believe there would be a need for a whitelist if application-level code such as Explorer and the Control Panels were improved, but let's assume I'm wrong and a whitelist is needed.

The default UAC setting in Windows 7 is: *"Don't notify me when I make changes to Windows settings."*

What this really means is: *"Don't notify me when Microsoft applications* require administrator rights."*

(*Or malware, but let's pretend the whitelist could be made secure for a moment.)

At the moment the whitelist, if you use it at all, is hardcoded and only for Microsoft's own code. Users cannot add third-party apps to the list. Users also cannot remove Microsoft apps from the list. This was confirmed by Microsoft Online Community Support in a microsoft.public.win32.programmer.kernel thread:

> *The change we made in Windows 7 default UAC settings is that any operation that is necessary to manage windows will not require an elevation - which in technical terms translates into a white list of trusted action / binaries which the user can make perform without UAC prompting from an elevation. This list does include windows file operations.*
>
> *You see a prompt in your File Manager program because your binary is not an inbox binary - i.e. not an executable which ships with windows. Hope that explains and clarifies. For security considerations, Windows 7 does not allow any 3rd party binary to be in the Windows trusted list. Therefore, your File Manager program still needs to handle the elevations.*

There are several other worthwhile comments in the short thread discussing user experience issues as well as the pitfalls of developers having to workaround the problem via their own solutions in order to provide parity with Microsoft's components.

The *"for security considerations"* part of the quote above is particularly ironic given it is this very whitelist which has opened a hole in UAC which allows silent elevation for any process that doesn't care about doing things

properly (or 100% reliably) and uses the backdoor.

If we are to have whitelisting then why is it limited to only Microsoft's code? And why can't the user remove particular Microsoft binaries from the whitelist?

- **Microsoft cannot argue that it's too risky to let users whitelist third-party code:**

  By creating so many obvious security flaws just in the way UAC has been abused in Windows 7, Microsoft have demonstrated -- without the need to look through history -- that they are no better at making secure application-level code than anyone else. Microsoft's user-mode executables have not *earned* special treatment when it comes to security or trust. The core OS should treat them with the same level of distrust as any other unelevated process making the same API calls.

  If people want to whitelist some of the Microsoft executables then that's up to them but there is nothing inherently trustworthy about them. (In addition, since they are well known and installed on every system, granting them whitelist status is more risky than doing the same to third-party or unbundled code.)

  Microsoft cannot argue that it's too risky to let users whitelist third-party code because Microsoft's own code has shown a disregard for preventing other processes hijacking its special elevation status.

- **Allowing users to add third-party code to the whitelist cannot make things less secure, only less annoying:**

  As it stands, if the whitelist is enabled at all, UAC prompts can be bypassed by anything that wants to. It is therefor hard to imagine how UAC could provide less security if the user was able to whitelist chosen, trusted and regularly used third-party programs.

  With the whitelist enabled UAC prompts are nothing more than security theater that protects against very little and only really gets in the way of legitimate programs and user-triggered actions, both of which the user will consent to when prompted. Seeing a UAC prompt when you do certain things in certain programs may give the *illusion* that UAC is still doing something towards your security but in reality it isn't doing much.

  The prompts in Vista, and in Windows 7 if you change to Always Prompt mode, made it very difficult for unelevated processes to gain elevation. In Windows 7's default mode you might as well turn the prompts off entirely.

  This also applies to non-bundled Microsoft applications such as the SysInternals tools. Why is it that Task Manager can create an elevated version of itself without prompting but if you use the superior Process Explorer then you have to confirm its launch via a prompt? Where is the security benefit here given that malicious code can bypass the prompt and Microsoft, so far, seem to say they don't care and have no intention of changing anything?

- **The inability to remove unused apps from the whitelist means security is being traded for nothing in return:**

A whitelist is about trading some security for some convenience. Every program on the whitelist adds to the attack surface of code that potentially can be exploited to gain admin rights. Every single item on the whitelist trades off a bit of security by making such an exploit more likely to be found.

If some of the whitelisted components are ones which the user rarely/never uses, what exactly is gained in return for this reduced security?

For example, I do not use Windows Explorer; I use another file manager. If I left the whitelist on then I would want to remove Explorer from. The only thing that might try to modify a protected folder on my machine using Explorer is a piece of software remote controlling Explorer either by sending mouse and keyboard events to it or, like my proof-of-concept program, by code-injection. I gain no convenience by having Explorer on the whitelist.

I also gain no convenience from having things like Calc.exe, Notepad.exe and MSPaint.exe on the COM elevation whitelist when they are programs which never actually use COM elevation unless something else has injected code into them!

- **Too many UAC prompts encourage users to turn off UAC; it doesn't matter which vendor causes the prompts:**

  People typically use software because it lets them do what they want, not because it's internally well designed or secure. If someone wants to use a program that triggers a lot of UAC prompts then they are more likely to turn off UAC than to switch programs.

  So, if the aim of the whitelist is to make UAC more bearable and discourage people from turning off prompts entirely then people need to be able to whitelist whichever applications they want. That way they will keep UAC on and UAC will still catch things done by programs they don't use often, or new programs. (Assuming, of course, that Microsoft manage to improve the UAC whitelist implementation so that malicious software cannot bypass the whole thing!)

  As said earlier, the excuse that UAC is supposed to make developers write better code does not wash when Microsoft's response to that challenge was to exempt their own code from UAC. If it's good for the goose then it's good for the gander.

Previous Windows 7 UAC issues

For completeness, a quick note about two other issues which were found in Windows 7's UAC shortly after the public beta was released. These were discovered and proved by Rafael Rivera Jr. and Long Zheng.

Microsoft initially seemed to disregard these two issues but, in the face of widespread criticism, apparently had a change of heart and said they'd be fix them in the Windows 7 release candidate.

(Let me stress that fixes for these two issues do not affect the code-injection issue that most of this page is about. The code-injection issue has so far had no substantial response from Microsoft.)

SendKeys:

Since the UAC control panel was not an elevated process/UI it was possible for a very simple piece of VBScript to open the control panel and then send keyboard events which moved the UAC slider and applied the change. Due to the elevated COM object which controls UAC being on the whitelist (list 2 in the section far above) there was no UAC prompt when the change was made. Microsoft have since promised that the UAC control panel will be in an elevated process/UI to prevent this from working. However, they themselves point out that many other control panels will still be in unelevated processes/UIs, using silently elevated COM objects. That means the SendKeys attack can still be used in lots of other places to mess with admin settings unless you set UAC to *Always Prompt*.

- Rafael's Within Windows: Malware can turn off UAC in Windows 7; "By design" says Microsoft
- I Started Something: Sacrificing security for usability: UAC security flaw in Windows 7 beta (with proof of concept code)
- I Started Something: Microsoft dismisses Windows 7 UAC security flaw, continues to insist it is "by design"
- I Started Something: Microsoft changes Windows 7 UAC control panel behavior to address security flaw

RunDll32.exe

Rafael discovered that RunDll32.exe was given auto-elevate (list 1) status. RunDll32.exe is a program which can, essentially, be told to run any other program. Because it auto-elevated it could be used to run anything with administrator rights without triggering a UAC prompt and without the launching process requiring administrator rights.

This is supposed to have been fixed in later internal betas but, as far as I know, there has not yet been any confirmation of this or what the fix is, exactly.

- Rafael's Within Windows: Windows 7 auto-elevation mistakes lets malware elevate freely, easily
- I Started Something: Second Windows 7 beta UAC security flaw: malware can silently self-elevate with default UAC policy

Deja Vu: Joanna Rutkowska and Vista

Although they're about Vista rather than Windows 7, it's also worth reading Joanna Rutkowska posts from February 2007 and the responses (or lack of) they generated, for deja vu if nothing else:

- Invisible Things: Running Vista Every Day!
- Invisible Things: Vista Security Model -- A Big Joke?
- Invisible Things: Confusion about the "Joke Post"

(Yep, UAC wasn't perfect back then either, but shouldn't Joanna's WM_KEYDOWN issue have been fixed rather than dismissed? If it required changes to accessibility software then fine, say so and make it an aim for the future, but don't say it's not important. Shouldn't the aim be to make UAC as secure as possible rather than to move in the opposite direction and make it a joke which insults third-party developers and tricks users into a false sense of security?)

To those saying, "but it requires code to get on the box"

Engineering Windows 7 blog:

I really don't buy the logic behind this statement on Microsoft's Engineering Windows 7 blog (which was written after the RunDll32.exe issue was found but before this page/issue was published):

> *It is important to look at the first step—if the first step is "first get code running on the machine" then nothing after that is material, whether it is changing settings or anything else.*

This isn't true. Yes, a problem which allows remote execution is far more important than one which allows local elevation. However, it very much is material whether or not a remote execution exploit can be turned into a remote **admin** code execution exploit. And there are other ways to get code to run on a box. Indeed, if all code already on the machine was immaterial then why do we have UAC prompts for anything in the first place? Are the prompts just there for show; just security theater?

Also, why do we have the Secure Desktop for UAC prompts if not to prevent running, supposedly-trusted applications from clicking their own UAC pompts and elevating without the user's permission?

(In both Windows 7 and Vista you can configure UAC to *"Elevate Without Prompting"*. That most keeps UAC on but automatically accepts all elevation requests. It still forces developers to write code that supports UAC, over-the-shoulder elevation for non-admin users, running minimal code at admin level, and so on. So the excuse that UAC prompts only exists to pester developers doesn't wash either.)

Either it is important to prevent random processes from elevating arbitrarily and silently, or it isn't:

- If it is important then Microsoft cannot dismiss this code-injection issue (or the SendKeys/RunDll32.exe issues which the E7 post was responding to).
- If it isn't important then the default UAC mode should get rid of *all* UAC elevation prompts for *all* applications, not just the Windows ones and the ones that bypass the prompts via exploits. In other words, run in the "UAC but without any prompts" mode that Vista already had.

I can see arguments for and against both bullet points.

Some people want to know when code they run requests full admin: More security and less convenience.

Some people implicitly trust any code they run and have it granted automatically: Less security and more convenience.

I'm cool with people in either camp. What I cannot accept is an argument for any mixture of the two: Less security and less convenience. It makes no sense to punish well-behaved apps by making them inflict UAC prompts on their users while ignoring apps that use exploits by allowing them to bypass the prompts (not to mention the malware that people *do* end up running, one way or another).

If prompts are important then they should not be something you can bypass with a trivial amount of code. On the other hand, if prompts are not important then let's get rid of them completely by default. Pick one.

You can't say that process elevation is worth protecting against when it's convenient (i.e. in places where Windows 7 already prompts by default and requires no code/design changes) while simultaneously saying that process

elevation isn't an issue in cases where it isn't convenient (convenient to the Windows 7 ship date, that is :-)). That's illogical, captain.

When the prompts are so easy to bypass by anything that wants to, having the prompts at all is pure security theater.

Secunia (via The Register):

Secunia said something similar in response to The Register's coverage of this code-injection issue:

> *This isn't a major issue; after all it requires that the user already downloaded some executable code and decided to run it. No matter which security features have been built into the operating system, then the user should never run code, which they don't trust in the first place. Untrusted code should only be run on dedicated test systems.*

First, it seems that Secunia's response is aimed at corporations who are unlikely to be affected by the problem at all because they tend to run everyone as non-admin anyway. General home users are unlikely to have "dedicated test systems" on which to test untrusted code.

Second, as for it not being a "major issue:" I don't dispute that there are more important issues, in particular remote code execution issues. Obviously. That does not mean that this issue is not worth worrying about, however. It can be used in conjunction with a remote code execution to worsen the damage. You do not wait until someone combines issues into a blended attack before you acknowledge them. **Malicious code *does* get on to people's machines.** That's a fact of life. People can be tricked into running things that don't look like executables, which is the rampant Conflicker Worm is doing as I type this. Even if we ignore trickery, exploits are regularly found which allow remote code execution though no action or fault of the user.

Once code gets on to a machine, one way or another, it's important to minimize what it can do. If someone finds a remote execution exploit in a non-admin process then it should not be easy for them to use it to gain full admin access. They can still do damage to an affected user's files but damaging the entire machine/OS should not be made easy, especially with methods which are completely silent to the user and do not arouse any suspicion (e.g. unexpected UAC prompts or visible user-interface remote-control). Especially when this problem was not that in Vista. The hole I have found allows non-admin code to elevate to full admin without the user seeing anything at all. The only reason the proof-of-concept videos show a UI is for demonstration purposes.

So while it isn't the most important security hole in the world -- and nobody ever said it was -- I find it strange that it could be considered unimportant... I guess it's all relative, though. (It is relatively unimportant compared to a lot of what Secunia look at, to be fair!) And, certainly, it's not important if you turn UAC up to its highest level, but the defaults need to be considered because that's what most people will use.

If you do consider it unimportant that, with the default settings, code can elevate itself freely once on the box then you must follow that to its logical conclusion. If local process elevation is a non-issue, and given that it's broken by default in Windows 7 anyway, then Windows 7 should default to *elevate without prompting* and not inflict UAC prompts on *any* programs. If people want to argue that then it's fine by me, provided I still have the option of setting UAC to *always prompt* on my own machine.

More to the point, there were better ways that Microsoft could have made UAC less annoying without opening such a large hole, as I explained in detail above.

To those saying, "but UAC isn't a security boundary"

That is a cop-out, IMO. Yes, there were ways things could piggy-back on or spoof elevations on Vista. Just because you can break some glass to enter a house doesn't mean you leave the doors unlocked, though.

First, I'm not aware of any UAC hole in Vista that could not be fixed. Changing the command prompt registry setting is often brought up as an example, so I'll use that: MS could fix that by making admin command prompts ignore the HKCU settings and only read from HKLM or an admin-ACL'd part of HKCU. Or by making either the UAC prompt itself or the admin process confirm to the user what it was about to run before it happens. (The OS needs to prove some way for elevated processes, whether by admin-approval or over-the-shoulder, to confirm what they ahve been asked to do before they go and do it.) Or maybe stuff like making Windows Defender alert you if the HKCU value is changed... There's a bunch of ways that problem could be addressed, as with every other I'm aware of. None of them are an excuse to ignore the code-injection issue in Windows7; rather, they are just other flaws in UAC (and how UAC is used) which should also be addressed.

(Following on from that, almost all of the piggy-back and spoofing issues apply eqally to over-the-shoulder elevation.)

Second, I think there's a significant difference between a program being able to gain elevation silently, the moment it wants to, and one which has to lie in wait for the user to do something it can exploit. Using the same example agan, if something wants admin on my box and modifies the command prompt settings in HKCU, it still has to wait for me to launch an elevated command prompt. It could be *days* before I trigger the action it is waiting for. Meanwhile it has to hope that I don't notice whatever changes it has made and, more importantly, that my anti-virus tool doesn't get an update which detects those changes. On the other hand, if something doesn't have to wait to get full admin access then it can *immediately* install a rootkit which means the anti-virus tool may not be able to detect it even after a signature update.

I mean, think about a typical single-user home or small office computer: What's important? Your data is important. You don't want something to delete or steal your private data, but full admin rights are not required for that so UAC is irrelevant to that discussion. It's also important to prevent your computer from being remote controlled, e.g. as a spam zombie, but once again admin rights are not required for that. Admin rights let things jump from one user to another but typically there only is one real user (possibly with multiple accounts) on the machines where UAC is used at all.

The main reason I can think of that a typical home user should care about software gaining full admin rights, over gaining any control to the system at all, is that with those rights it can hide itself deeply and *never* be detected (or only be detected a very long time in the future), or maybe install low-level stuff like keyboard hooks to sniff passwords. In cases like those I think it does matter how long something has to wait to gain that access. If something has got on my machine through a remote execution exploit in Adobe Flash (or whatever) and I've had my data compromised, or become part of a botnet, then that really sucks. But it'll suck even more if I don't realise it's going on for six months instead of a day. I'm not saying a solid UAC will *always* prevent that from happening

but it could prevent it in some cases and should do a hell of a lot more than a UAC which can be trivially bypassed.

Third, what *is* UAC actually supposed to do? The reality is that you'll get a different answer from different people. Sometimes if you ask the question in different contexts you will get a different answer from *the same people*. Because UAC is a fuzzy thing that's supposed to kinda help make things sort-of more secure and also to kinda force developers to do some stuff differently, without breaking anything or annoying any one too much, and it's all a bit undefined, and Microsoft now have one set of rules for themselves and another set of rules for everyone else... The realiy is that UAC does whatever it's most convenient to tell you it does at any particular time, and goals that UAC had in Vista have apparently been reduced -- or removed entirely by revisionism -- in Windows 7. I'm asking for UAC's purpose to be properly and explicitly defined and for Microsoft to then consistently and fairly stand behind whatever that is, for both their own software and third-paty software. (And if getting people to run as standard user is *really* the aim of UAC, and not just a BS excuse to ignore the problems people raise, then it'd be nice if Microsoft actually made doing that not a giant pain in the behind that is even worse than the UAC prompts they know people don't like.) There are more of my arguments along these lines throughout this page but I'll repeat one more example of the schizophrenia: Why do we have a Secure Desktop for UAC prompts if it doesn't matter that applications can silently elevate via a backdoor?

To those saying, "but it's only a beta"

Of course Windows 7 is only a beta. All of us trying to expose the problems introduced by the UAC changes hope that Microsoft will improve things before Windows 7 ships. However:

- It took a lot of people making a lot of fuss to make Microsoft take notice of the first two UAC flaws.
- Ignoring the Web2.0 abuse of the phrase, the point of a beta version is to allow testing and feedback. We are testing and providing feedback on UAC. Too many people seem to think that "beta" means "a way for me to get a shiny new piece of software early." Stop it.
- Like it or not, Microsoft tend to do a terrible job at fixing things in a reasonable amount of time. There are many examples but for a particularly bad one, consider the file-corruption bug in Windows Home Server which was known about when the OS was in beta and not fixed until a year after it was released to retail.
- Security design and code is best done sooner rather than later. It is very difficult to take an insecure design and make it secure. The sooner problems start to be addressed the better.
- Microsoft have said there will not be another public beta of Windows 7. The next public release will be RC1. Windows 7 is clearly on an aggressive release schedule.
- There is no point waiting until the retail release of Windows 7 to complain. Getting Microsoft to change anything in Windows after a version has shipped is like pulling teeth. By then it is too late. It may *already* be too late but we live in hope.
- Update: Windows 7 RC1 (build 7100) is out now and nothing has changed in this area. (A lot of other stuff has changed and it's a better version of Windows, but the UAC/injection stuff still works like it used to.) It now seems even more unlikely that anything will be changed in time for the RTM, especially as MS still haven't replied to my offers to give them full details.

Quick response to a couple of newer things:

I should work these into the main page but don't have the time/energy, so I'll put them here for now.

- **Inside Windows 7 User Account Control by Mark Russinovich in the July 2009 TechNet Magazine:**

  Some good technical information there explaining how the manifests etc. work.

  IMO, though, it contains the usual contradictory reasoning about what UAC prompts are for and not for that is coming out of Microsoft. I have a lot of respect for Mark Russinovich and he obviously knows more about Windows than I do but I disagree with much of what he says on this matter.

  "From the perspective of malware, Windows 7's default mode is no more or less secure than the Always Notify mode ("Vista mode"), and malware that assumes administrative rights will still break when run in Windows 7's default mode."

  Following that argument through, the "silently elevate" setting is no more or less secure as well, so why aren't Microsoft using that as the default? Why are they making third party apps (as well as Mark's own apps!) look bad and annoy users if it provides no benefit?

  If the UAC *prompts* serve no purpose for admin users, except to force vendors to change their apps, then why did MS fail to change their own apps and why did they make themselves exempt from the prompts but not allow others to do the same?

  He says the code-injection method isn't trivial, yet it only took me a couple of days to research & write from scratch and I'm an app developer, not a hacker/cracker. Perhaps not trivial but not time consuming, either. It'll certainly be trivial once the source code is released (by me or by or someone else who bothered to write something similar) as you just call an function with the command you want to run and it happens.

  However, I do agree that it's not something legitimate software should rely upon. (And yet, if third parties want to offer the same user experience which Microsoft allow for their own software, using this method is the least-work alternative. That sucks. The other alternatives are convincing users that the UAC prompts are pointless and that they should turn them -- but not UAC itself -- off, or writing a service which runs as admin and accepts requests for work that requires admin rights, which is a lot more effort, difficult to secure and wasteful of resources. That sucks, too!)

  If the prompts provide no security for admin accounts, and you don't care if the prompts can be bypassed for admin accounts, then turn the prompts off for admin accounts for all apps. You can still keep UAC and make it so that apps have to ask for administrator rights to get them (so that old malware fails by default and so that apps support elevatrion from standard user account). What is the benefit of the prompts for admin users if Microsoft do not want to make the improvements required to turn them into a security feature (and indeed if MS are working in exactly the opposite direction)? There is none. So turn off the prompts for admin users (without turning UAC itself off).

  And, as I've said already, if MS think people are going to switch to standard user accounts and type a password for every elevation -- while also making changes to Windows 7 which admit that the less-annoying UAC button-click prompts for admin accounts in Vista were too annoying for many people to use -- then they must be smoking high-strength crack.

"We understand the aggravation, and there might be a legitimate reason that those applications can't run without administrative rights, but the risk is too high that developers will avoid fixing their code to work with standard user rights."

That comment seems disingenuous to me. Making *do not prompt* the default UAC level (at least for COM elevation) would not allow developers to avoid fixing their code. Developers would still need to deal with the UAC API and request elevation in a way which would work under standard user accounts. The only difference would be that the requests would not trigger a prompt. That's how it works for Explorer in Windows 7; why can't it work that way for everyone?

It makes no sense. The argument seems to be that, simulataneously, silent elevation from limited admin to full admin is too dangerous to allow third-parties to use and yet it's no problem at all -- not even worth asking for the full details of! -- if it can be triggered via a workaround. If the UAC prompts are so useless then switch them off.

The current Windows 7 defaults are anti-competitive security theater. Microsoft need to stand up and admit it.

The discussion after Vista should have been about how to make it harder to spoof UAC prompts -- e.g. by the protected-side code telling you more information about what it had been asked to run by the unprotected-side code, and closing off holes like the HKCU cmd.exe registry settings -- and about how to design software to minimize the number of prompts -- and retarded prompts-about-prompts in particular -- per logical operation. (Or you could give up and turn off the prompts for admin users in all software, but that still doesn't make UAC stop sucking for the barely-existent "standard user account that might elevate" case who seems to be the excuse for the mess. I'm not saying there aren't a lot of standard-user accounts out there in businesses but I am saying that very few of them use UAC to elevate to administrator, and in the home it's similar because it's such a pain to type the password for every UAC prompt, which MS know and admit by their changes to Win7's admin accounts.) Instead the discussion is this waste of time where we're arguing about the validity of MS adding a hack just for themselves which makes things even worse. Sigh.

Mark also fails to acknowledge that many UAC prompt spoofs which would work with an admin account on Vista would work just as well with a standard user account on Vista or Windows 7. He's pointing out a flaw in the old system as an excuse for the new system's design, yet that flaw still exists in the new system.

I have a lot of respect for Mark but I think his reasoning is inconsistent here. It reads more like an after-the-fact excuse for a poorly designed, unfair and schizophrenic system rather than a solid set of design goals which producted a system that works as well as it could.

- **PCMag: [Bogus Claims of Win7 UAC Vulnerabilities]**

This blog post was published in a couple of places. I'll re-post my response here (which so far has not been replied to):

Larry,

You seem to fall into the "UAC is pointless" side of the camp, which is fine. Argue for it [the prompts] to be turned off completely in that case. That has not happened, though, so you should be just as annoyed by the Windows 7 changes as anyone else as they have made UAC *even more pointless*.

You also seem to be saying it doesn't matter because people should be running as standard user, and not doing admin things at all. While it may be true that they should be, it is not true that they are. It is unlikely to ever be true unless Microsoft significantly improve the experience of standard users, which they have not.

The facts are:

- Administrator is the default. You have to go out of your way to change it.
- People *do* do Administrator tasks. If they didn't then UAC would not have annoyed anyone in the first place and the Windows 7 changes would never have happened. I doubt that will ever change.
- People were too annoyed by the UAC prompts on Vista when running as an Administrator and Microsoft clearly realised this (but their solution is terrible). (PERSONALLY, I thought UAC in Vista was fine, but perhaps that's because I don't use Windows Explorer which is the reason for a lot of UAC's irritation.)
- If people were annoyed by the UAC prompts on Vista there is absolutely no way they will put up with the standard user prompts in Windows 7 (which are essentially the same as they were in Vista, and much *worse* than the admin prompts in Vista which caused the UAC backlash in the first place).
- Windows 7's changes have made it *more* tempting to run as an administrator, not less. They're a step in the wrong direction if your aim is for everyone to run as standard user.

"Bear in mind that you and Davidson are arguing for the Vista-style UAC."

I am not so much arguing for Vista-style UAC as against Windows 7-style UAC. You're creating a false dichotomy by equating the two positions.

"Isn't the consensus that this was a flop in the market? Do you have any constructive advice for Microsoft?"

Yes, plenty. Did you actually read my page? :) I know it's long and rambling and really needs editing, but it is full of examples of how UAC could have been done better without the Windows 7 mess.

Not just theoretical examples, either. Many of the examples are implemented in a commercial app where the UAC-related code was written back in 2007 before anyone had heard of Windows 7.

As I've said in several places, people asked for change but nobody asked for *this*.
