“Fileless” UAC Bypass Using eventvwr.exe and Registry Hijacking

Published: 2016-08-15 · Archived: 2026-04-05 18:57:37 UTC

After digging into Windows 10 and discovering a rather interesting method for bypassing user account control, I decided to spend a little more time investigating other potential techniques for getting around UAC. Currently, there are a couple of public UAC bypass techniques, most of which require a privileged file copy using the IFileOperation COM object or WUSA extraction (Windows 7) to take advantage of a DLL hijack in a protected system location. All of these techniques require dropping a file to disk (for example, placing a DLL on disk to perform a DLL hijack). You can take a look at some of these public techniques here (by @hfiref0x). The technique covered in this post differs from the other public methods and provides a useful new technique that does not rely on a privileged file copy, code injection, or placing a traditional file on disk (such as a DLL). This technique has been tested on Windows 7 and Windows 10, but is expected to work on all versions of Windows that implement UAC.

As I mentioned in my last post on bypassing UAC using Disk Cleanup, a common technique used to investigate loading behavior on Windows is to use SysInternals Process Monitor to analyze how a process behaves when executed. While digging into the Windows Event Log with ProcMon opened, I noticed that eventvwr.exe was executing some registry queries against the HKEY_CURRENT_USER hive as a high integrity process.

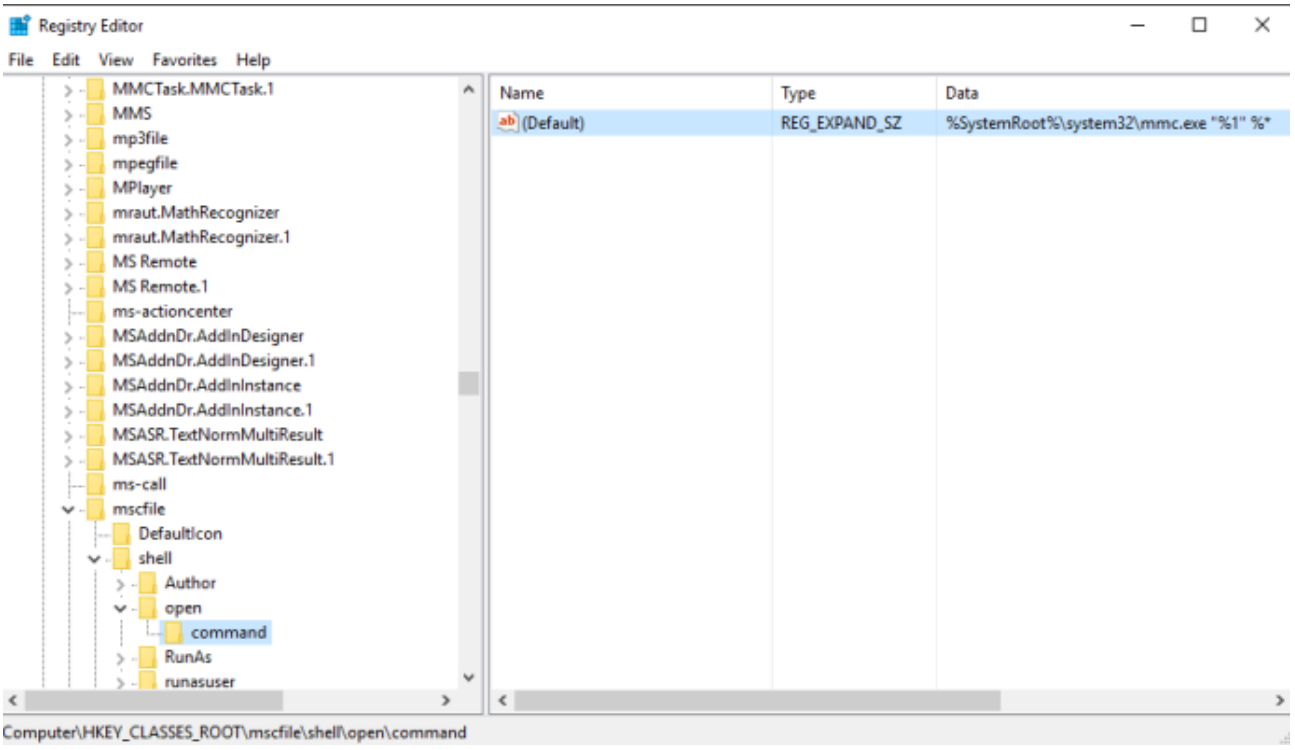
Before diving in too far, it is important to understand what the HKEY_CLASSES_ROOT (HKCR) and HKEY_CURRENT_USER (HKCU) registry hives are and how they interact. The HKCR hive is simply a combination of HKLM:\Software\Classes and HKCU:\Software\Classes. You can read more about HKCR and why the HKLM and HKCU hives are merged here. Since these hives are merged, you can often hijack keys in HKCR:\ by creating them in HKCU:\Software\Classes. Since this relationship exists between these 2 hives, any elevated process that interacts with both HKCU and HKCR in succession are particularly interesting since you are able to tamper with values in HKCU. As a normal user, you have write access to keys in HKCU; if an elevated process interacts with keys you are able to manipulate, you can potentially interfere with actions a high-integrity process is attempting to perform.

Now, as some of you may know, there are some Microsoft signed binaries that auto-elevate due to their manifest. You can read more about these binaries and the manifests here. By using the SysInternals tool “sigcheck”, I verified that “eventvwr.exe” auto-elevates due to its manifest:

```
<description>Event Viewer Snapin Launcher</description>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
        <requestedPrivileges>
            <requestedExecutionLevel
                level="highestAvailable"
                uiAccess="false"
            />
        </requestedPrivileges>
    </security>
</trustInfo>
<asmv3:application>
   <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
        <autoElevate>true</autoElevate>
   </asmv3:windowsSettings>
</asmv3:application>
</assembly>
```

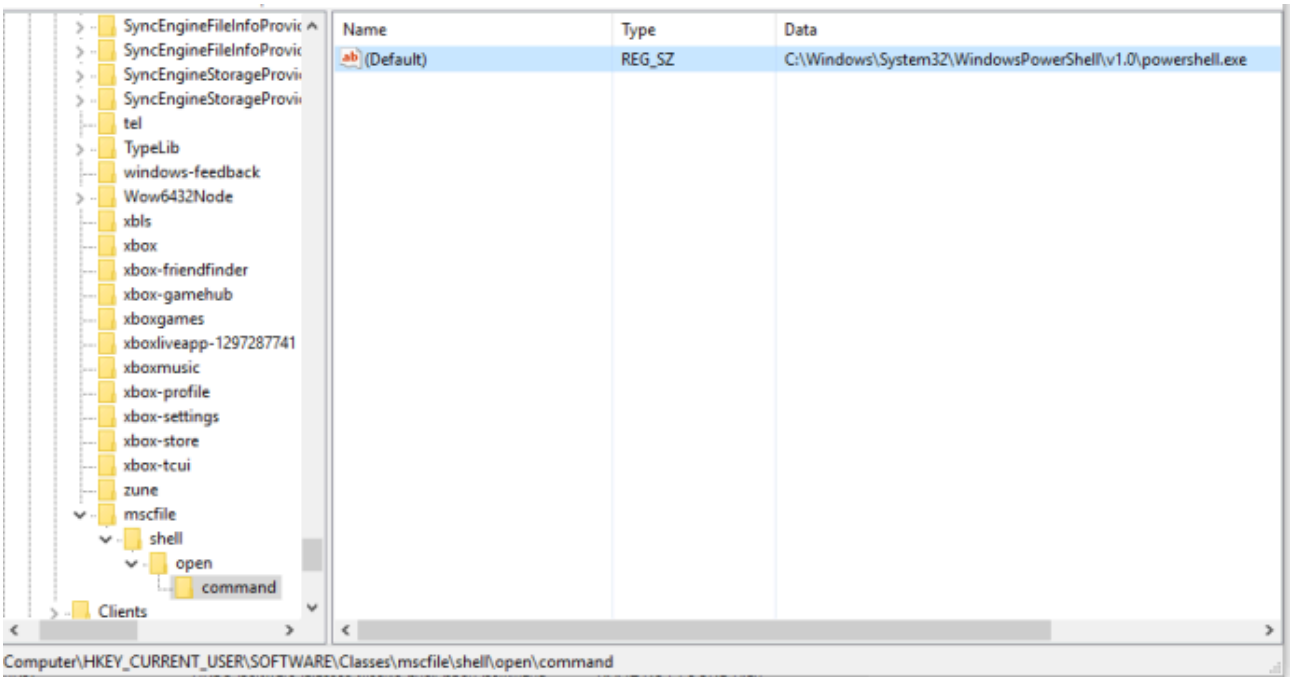
While digging deeper into the ProcMon output, I noticed that “eventvwr.exe” was interacting with **HKCU\Software\Classes\mscfile\shell\open\command**, which resulted in a “NAME NOT FOUND” result. Shortly after, eventvwr.exe was seen interacting with the key **HKCR\mscfile\shell\open\command**. Looking at **HKCR\mscfile\shell\open\command**, I noticed that the default value was set to call mmc.exe (Microsoft Management Console), the program responsible for loading Management Snap-Ins:



As mentioned above, calls to HKEY_CURRENT_USER (or HKCU) from a high integrity process are particularly interesting. This often means that an elevated process is somehow interacting with a registry location that a medium integrity process can tamper with. In this case, it was observed that “eventvwr.exe” was querying **HKCU\Software\Classes\mscfile\shell\open\command** before **HKCR\mscfile\shell\open\command**. Since the HKCU value returned with “NAME NOT FOUND”, the elevated process queried the HKCR location:

eventvwr.exe	3120	RegOpenKey	HKCU\Software\Classes\mscfile\shell\open\command	NAME NOT FOUND	High
eventvwr.exe	3120	RegQueryKey	HKCR\mscfile\shell\open	SUCCESS	High
eventvwr.exe	3120	RegOpenKey	HKCR\mscfile\shell\open\command	SUCCESS	High
eventvwr.exe	3120	RegQueryKey	HKCR\mscfile\shell\open\command	SUCCESS	High
eventvwr.exe	3120	RegQueryKey	HKCR\mscfile\shell\open\command	SUCCESS	High

From the output, it appears that “eventvwr.exe”, as a high integrity process, queries both HKCU and HKCR registry hives to start mmc.exe. After mmc.exe starts, it opens eventvwr.msc, which is a Microsoft Saved Console file, causing the Event Viewer to be displayed. This makes sense due to the fact that the Microsoft Management Console (mmc.exe) loads Microsoft Saved Console files (.msc). You can read more about the Microsoft Management Console and the corresponding Microsoft Saved Console files here.
With this information, I decided to create the registry structure needed for “eventvwr.exe” to successfully query the HKCU location instead of the HKCR location. Since the (Default) value located in **HKCR\mscfile\shell\open\command** contained an executable, I decided to simply replace the executable with powershell.exe:



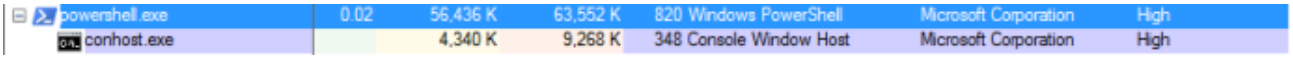
When starting “eventvwr.exe”, I noticed that is successfully queried/opened **HKCU\Software\Classes\mscfile\shell\open\command:**

eventvwr.exe	4756	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS	High
eventvwr.exe	4756	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS	High
eventvwr.exe	4756	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open	SUCCESS	High
eventvwr.exe	4756	RegOpenKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS	High
eventvwr.exe	4756	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS	High
eventvwr.exe	4756	RegQueryKey	HKCU\Software\Classes\mscfile\shell\open\command	SUCCESS	High

This action effectively replaced the expected “mmc.exe” value with our new value: “powershell.exe”. As the process continued, I observed that it ended up starting “powershell.exe” instead of “mmc.exe”:

eventvwr.exe	4756	Process Create	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	SUCCESS	High
powershell.exe	820	Process Start		SUCCESS	High
powershell.exe	820	Thread Create		SUCCESS	High

Looking at Process Explorer, I was able to confirm that powershell.exe was indeed running as high integrity:



Due to the fact that I was able to hijack the process being started, it is possible to simply execute whatever malicious PowerShell script/command you wish. This means that code execution has been achieved in a high integrity process (bypassing UAC) without dropping a DLL or other file down to the file system. This significantly reduces the risk to the attacker because they aren't placing a traditional file on the file system that can be caught by AV/HIPS or forensically identified later.

To demonstrate this attack, Matt Graeber (@mattifestation) and I constructed a PowerShell script that, when executed on a system, will create the required registry entry in the current user's hive (**HKCU\Software\Classes\mscfile\shell\open\command**), set the default value to whatever you pass via the -Command parameter, run "eventvwr.exe" and then cleanup the registry entry.

You can find the script here: https://github.com/enigma0x3/Misc-PowerShell-Stuff/blob/master/Invoke-EventVwrBypass.ps1

Within the script, we have provided an example command. This particular command uses PowerShell to write out "Is Elevated: True" to C:\UACBypassTest. This will demonstrate that the command has executed has a high integrity process due to the fact that "Is Elevated" equated to "True" and the text file it outputs is being written to a directory that a medium integrity process is not allowed to write to.

This technique differs from the other public techniques by having a few handy benefits:

1. This technique does not require dropping a traditional file to the file system. Most (if not all) public UAC bypasses currently require dropping a file (typically a DLL) to the file system. Doing so increases the risk of the attacker getting caught. Since this technique doesn't drop a traditional file, that extra risk to the attacker is mitigated.
2. This technique does not require any process injection, meaning the attack won't get flagged by security solutions that monitor for this type of behavior.
3. There is no privileged file copy required. Most UAC bypasses require some sort of privileged file copy in order to get a malicious DLL into a secure location to setup a DLL hijack. Since it is possible to replace what executable "eventvwr.exe" starts to load the required Snap-in, it is possible to simply use an existing, trusted Microsoft binary to execute code in memory instead.

This particular technique can be remediated or fixed by setting the UAC level to "Always Notify" or by removing the current user from the Local Administrators group. Further, if you would like to monitor for this attack, you could utilize methods/signatures to look for and alert on new registry entries in **HKCU\Software\Classes**.
