COM Elevation Moniker DLL Hijacking Privilege Escalation

Author: Borja Gómez

Country: Spain

E-Mail: kub0x@elhacker.net

Date of discovery: 16/09/2014

Estimated Microsoft employee,

I leave here the technical details as the requirements, exploitation and contra measures of the discussed vulnerability. **The general information as the requirements are the same in both exploits.**

General Information:

Exploit Type: Local privilege escalation in user-mode.

Tested on:

- Windows 7 Professional 32 bits & 64 bits

- Windows 8.1 Professional 64 bits

***** May affect all Windows 7 OS families, same for Windows 8.0/8.1 & Vista.

Requirements:

- User belongs to the administrators group.

- UAC is enabled in any option, except the highest one " ".

Additional Software:

My exploit has been coded on Visual Studio 2013 so you need to install the latest C++ Redistributable in order to execute the POC, but this could be avoided including essential libraries in the executable.

- **Microsoft Visual C++ 2012 Redistributable (x86) 11.0.61030**

- **Microsoft Visual C++ 2012 Redistributable (x86) 11.0.61030**

- **Microsoft Visual C++ 2013 Redistributable (x86) 12.0.21005**

- **Microsoft Visual C++ 2013 Redistributable (x64) 12.0.21005**

Vulnerability details:

The flaw occurs when we replace a legit DLL of an auto-elevated process with ours (AKA DLL Hijacking) using an auto-elevated COM object from a signed entity that allows the COM object's execution. In Windows 8/8.1 it bypasses the white list specified to fix this the known flaw, so it reopens the security breach allowing malicious code to escalate to SYSTEM.

Also **there is a fresh discovered attack here, we don't need Sysprep.exe** to execute our code.

I will also explain this approach later.

What does this means?

This attack was previously announced and fixed by Microsoft. In Windows 8.1 Pro Sysprep.exe hardcodes a path in its manifest to known DLLs that are load in runtime, so this is supposed to ensure DLL Hijacking protection, but it does not, so the fix is useless.

Bypassing sysprep's DLL Whitelist in Windows 8/8.1.

If you debug Sysprep.exe and list its loaded dependencies at runtime then you will see a DLL called SHCore.DLL. You can place yours (also named SHCore.DLL) in the same directory as Sysprep.exe.

When Sysprep.exe is executed, it will load our malicious DLL and we will be able to execute any process as SYSTEM, register privileged task, load a driver, just whatever you want to do.

What about Windows 7?

Windows 7 doesn't provide a whitelist for sysprep.exe so I have picked up CRYPTBASE.DLL for completing the DLL Hijacking.

What is the other approach you mentioned?

Also I have discovered that you do not need to target Sysprep.exe for doing this task. You can copy an auto-elevated process to C:\Windows\System32\Sysprep\. Also our malicious DLL named "CRYPTBASE.DLL" is copied into there. If we open the copied executable, it will load our DLL in Admin security context because it's an auto-elevated process.

Example: copy our DLL and msconfig.exe into /Sysprep/ directory. Execute msconfig.exe and it will load our DLL.

But, how do you copy the DLL to a protected Directory?

IFileOperation's **Elevation COM Moniker** could be used to run actions as admin. To copy files we only need to specify the source file, destination folder and the name that the copied DLL would receive.

Additional info.

IFileOperation interface is located at the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{3ad05575-8857-4850-9277-11b85bdb8e09}

Also includes a sub key that specifies that the COM object can be elevated, so I take profit about how this behaves.

What about the left clues?

Well, IFIleOperation also supports file deletion, so the items that were copied are deleted in order to clean our mess.

Remarks:

- **IFileOperation::PerformOperation()** won't prompt UAC when user belongs to administrator group.
- Also, UAC won't prompt when executing an auto-elevated process under a user account that belongs to the administrator groups.

Technical details:

With **CoGetObject** we can bind the COM Elevation Moniker with the **IFileOperation** object. IFileOperation interface is allowed to copy, move and delete files, so what we do is to place a DLL into Sysprep's protected SYSTEM and then execute sysprep.exe. The other explained approach is to copy our DLL and an auto-elevated process to Sysprep's directory, and execute the copied executable.

NOTE: I use **rundll32** to load my malicious DLL. This has a big **advantage**. Rundll32 needs to find an **EntryPoint**, **DllMain** also is called. When **EntryPoint** is called we know that we are inside Rundll32, so we only copy ourselves into Sysprep directory. When sysprep/others are executed, our **DllMain** is called, creating a cmd running under **SYSTEM**, but it doesn't execute the EntryPoint, so we don't re-copy ourselves. Also rundll32 is a secure and trusted source of execution.

Remarks:

COM Elevation moniker must be requested from a Thread. The fact is that the code will run in the EntryPoint so COM won't initialize properly.

I use function pointers for the WInAPI. This was added just for fun when I finished the PoC code.

So we begin initializing COM and creating the mentioned interfaces:

```
if (SUCCEEDED(pfnCoInitializeEx(NULL, COINIT_APARTMENTTHREADED)))
{
	IFileOperation *pfo = NULL;
	IShellItem
		*psiFrom = NULL,
		*psiTo = NULL,
		*psiDelete = NULL;

	BIND_OPTS3 bopts;
	ZeroMemory(&bopts, sizeof(bopts));
	bopts.cbStruct = sizeof(bopts);
	bopts.hwnd = NULL;
	bopts.grfMode = STGM_READWRITE;
	bopts.dwClassContext = CLSCTX_LOCAL_SERVER;

	if
(SUCCEEDED(pfnCoGetObject(L"Elevation:Administrator!new:{3ad05575-8857-
4850-9277-11b85bdb8e09}", &bopts, IID_PPV_ARGS(&pfo))))
	if (SUCCEEDED(pfo->SetOperationFlags(FOF_NO_UI |
FOFX_REQUIREELEVATION | FOFX_NOCOPYHOOKS)))

	[..]
```

SetOperationFlags will indicate that UAC won't prompt, also the dialog won't be shown to the user, so the copy process will be done in silence.

Then we instantiate 2 IShellItem COM interfaces. These are used to translate the source file and destination paths into objects that will be managed by the IFileOperation object in the copying process.

```
if (SUCCEEDED(pfnSHCreateItemFromParsingName(pszSrcItem, NULL,
IID_PPV_ARGS(&psiFrom))))
if (SUCCEEDED(pfnSHCreateItemFromParsingName(pszDest, NULL,
IID_PPV_ARGS(&psiTo))))

[..]
```

Now we attempt to copy our file into the destination path:

```
if (SUCCEEDED(pfo->CopyItem(psiFrom, psiTo, pszNewName, NULL)))
if (SUCCEEDED(pfo->PerformOperations())){

[..]
```

Okay our malicious DLL has been copied into Sysprep directory. The next step is to execute sysprep.exe/others. That will result in the execution of our DLL.

```
SHELLEXECUTEINFOW shinf = { 0 };
shinf.cbSize = sizeof(shinf);
shinf.fMask = SEE_MASK_NOCLOSEPROCESS;
shinf.lpFile = procPath.c_str(); //Sysprep or msconfig
shinf.lpDirectory = pszDest;
shinf.nShow = SW_HIDE;
if (pfnShellExecuteEx(&shinf) && shinf.hProcess){
	pfnWaitForSingleObject(shinf.hProcess, INFINITE);
	pfnCloseHandle(shinf.hProcess);
}
```

And we are done. The files will be cleaned and my DLL will spawn a cmd as SYSTEM impersonating Winlogon.exe token. I can do this because my DLL is loaded by an auto-

elevated process. In this PoC, like I said, I use msconfig.exe and sysprep.exe creating two good choices. Also whitelist of sysprep is bypassed in Windows 8.1, so we can enjoy both features too.

The **code is located at "COM PoC" directory**, inside the .rar that I provided. Please read the **README** as it specifies the documentation and procedures that have been taken.

There are two executable files, one for each processor architecture (32/64 bits).

Execute the POC code fulfilling the requirements and **without** running as an administrator.

In this PoC I spawn a CMD running under SYSTEM account using DLL Hijacking and executing sysprep.exe or msconfig.exe. You can set **more vulnerable processes** like "**SystemPropertiesComputerName.exe**".

PoC Remarks:

The entry point of my DLL in x86 is **_EntryPoint@16** and in x64 is **EntryPoint**. Rundll32 needs the entry point's name as input, so be careful.

When you introduce '1' as input parameter, **sysprep.exe** would be used. It bypasses sysprep's know DLL list white list in Windows 8.1.

When you introduce '2' as input parameter, **msconfig.exe** will be copied to /Sysprep/ directory and will be executed there.

NOTE: We could spawn a rundll32 as SYSTEM, so our DLL could will execute at same level.

Vector Attacks:

Binding our DLL as a resource in an .exe. The .exe will spawn the DLL in Rundll32 and you know the rest. This can be included in driven-by attacks or in a virus spreader that attaches to legitimate executables.

Contra measures:

- White listing DLLs. I know that KnownDlls path is stored at the registry, and those DLLs cannot be targeted to DLL Hijacking attacks. Auto-elevated processes are very sensitive, Sysprep.exe uses DLL white listing in its manifest, but SHCORE.DLL is missing. Please add a DLL white list in each auto-elevated processes manifest. You also could create a special registry key with the DLLs to protect, or just modify KnownDlls key to avoid totally the impersonation of those DLLs.

- Run under a **standard user account**. This means that the users does not belong to the administrators group, so UAC prompts and ask for the administrator credentials.

- Mark "**Always Notify**" option in the User Access Control Settings Panel. Is the highest option, so will prompt UAC window for each task that involves elevation, including the execution of auto-elevated processes.

That's all. Please read the README before reading the PoC code. I hope it clears your doubts.

Thanks for reading and I hope you answer soon ☺

Best regards,

Borja.