**Description of the vulnerability**

The task scheduler service has an alpc endpoint, supporting the method “SchRpcSetSecurity”.

The prototype looks like this:

long \_SchRpcSetSecurity(

[in][string] wchar\_t\* arg\_1, //Task name

[in][string] wchar\_t\* arg\_2, //Security Descriptor string

[in]long arg\_3);

Tasks created by the task scheduler will create a corresponding folder/file in c:\windows\system32\tasks. This function seems to be designed to write the DACL of tasks located there, and will do so while impersonating. However, for some reason it will also check if a .job file exists under c:\windows\tasks and try to set the DACL while not impersonating. Since a user, and even a user belonging to the guests group can create files in this folder, we can simply create a hardlink to another file (all we need is read access). Because of the hardlink, we can let the task scheduler write an arbitrary DACL (see second parameter of SchRpcSetSecurity) to a file of our choosing.

So any file that we have read access over as a user and that system has the write DACL permission for, we can pivot into full control and overwrite it.

**Exploitation**

This vulnerability gives us a really strong primitive! The main issue is that in a default installation, a lot of critical files can only be modified by trusted installer (not even system).

I have included a powershell script to enumerate files you can take control over. Just run ./enumerate.ps1 > output.txt

There are still a lot of targets (Also, you can take control of things in program files, and if the target is shared by admins/other users, they potentially end up running programs you can overwrite)!

The second issue is, while we can take control of all these files, writing to them is often blocked since many dlls are already loaded somewhere, thus will trigger a sharing violation.

There may be file types besides .dll’s that would be a better target.

C:\Windows\System32\DriverStore\FileRepository\prnms003.inf\_amd64\_4592475aca2acf83\Amd64\printconfig.dll (Folder name might vary.. but I accounted for that in my PoC)

This was one of the dlls I got a hit on. It seems to relate to the xps printer, and is not loaded into the print spooler service by default (it might happen that its loaded already.. but more often its not it seems).

So if we start a print job using the xps printer, it will load this dll, which we can overwrite with our vulnerability.

This hijacking vector can be easily swapped out for something better. I can try to find better alternatives.. just let me know.

(edit: On an old laptop which has been running win10 for a few years.. there where two prnms003.inf\_amd64\_\* folders. The newer version didn’t delete the older one, and there is no garantuee that FindFirstFile as used in the PoC will target the newest.. so you may want to expand this and add FindNextFile to overwrite printconfig.dll in any hits you get on prnms003.inf\_amd64\_\*, or check the last modified attribute and only overwrite the newest.)

**Payload and extra considerations**

The payload (which is used to overwrite our target dll/file), is attached as a resource in the visual studio project, you can change it there or change exploit.dll under the folder resource.

Also in code will be the following line:

HMODULE mod = GetModuleHandle(L"ALPC-TaskSched-LPE");

This has to reflect the name of the final dll, or the PoC won’t work, since it won’t know where to fetch the payload resource.

Should the PoC not work, check spoolsv.exe in process explorer and make sure printconfig.dll is not already loaded. It should not happen often.. but it might for whatever reason..

Included a new video demo, you should see those results while reproducing (tested on 64-bit, you need some extra work for x86 support, since the payload is x64, and you need to overwrite the x86 version of printconfig.dll instead too).

Again, if anything needs changing, needs extra features, or if you need a more reliable hijacking vector, just let me know.