HISPASEC SISTEMAS
SEGURIDAD Y TECNOLOGÍAS
DE LA INFORMACIÓN

Whitepaper

Windows LongPaths: extended-length path

Marcin “Icewall” Noga
martin@hispasec.com

Index

Hispasec Sistemas S.L.

Avda Juan López Peñalver, 17
Edificio Centro de Empresas CEPTA
Parque Tecnológico de Andalucía
29590 Campanillas (Málaga)

Telf: (+34) 902 161 025
Fax: (+34) 952 028 694

Información General
info@hispasec.com

Comercial
comercial@hispasec.com

www.hispasec.com

1. Basic information about paths

1.1. Common lengths of paths and its extended version

Most programmers are used to use a path with maximal length limited to MAX_PATH, this is 260 characters. Microsoft documentation **Naming Files, Paths, and Namespaces** states:

maximal count of signs in a path can reach 32,767 characters

Is this number familiar? No? Let us do some maths:

32,767 unicode characters are 65,534 bytes

But why exactly 65,534 is the path length limitation? The answer is obvious when we have a look at the FILE_OBJECT structure definition which is used to represent a file, directory, device or volume object instance.

The most interesting field in this structure for our purposes is:

UNICODE_STRING FileName

Let us look at the UNICODE_STRING structure definition.

```
typedef struct UNICODE_STRING {
        USHORT  Length;          /* bytes */
        USHORT  MaximumLength;   /* bytes */
        PWSTR   Buffer;
} UNICODE_STRING,*PUNICODE_STRING;
```

Now everything should be clear. Paths are limited to 65,534 bytes because the field's path length is of type USHORT, the maximal value of it being 2^{32} -1 = 65,535.

1.2. Bypass MAX_PATH limitation with the \\?\ prefix

According to the **Maximum Path Length Limitation** documentation, a path can be extended to 32,767 characters but the user needs to add "\\?\" at the beginning of it. For example:

\\?\C:\Path\longer\than\max_path

Many of the readers will want to know *where* exactly is the routine responsible for checking the MAX_PATH limitation located and the entire mechanism related to it.

2. Path length limitation analysis

Reading the code of API functions such as CreateFile or CreateDirectory we can notice a couple of common calls to other APIs, the most important for us is **RtlDosPathNameToNtPathName_U.**

We can find the implementation of this function in ntdll.dll

```
NTSYSAPI
BOOLEAN
NTAPI
RtlDosPathNameToNtPathName_U(
    __in PCWSTR DosFileName,
    __out PUNICODE_STRING NtFileName,
    __out_opt PWSTR *FilePart,
    __reserved PVOID Reserved
    );

.text:7C9142F5                 public _RtlDosPathNameToNtPathName_U@16
.text:7C9142F5 _RtlDosPathNameToNtPathName_U@16 proc near
.text:7C9142F5                                         ; CODE XREF:
.text:7C9142F5                                         ;
LdrpMapDll(x,x,x,x,x,x)+7B8p ...
.text:7C9142F5
.text:7C9142F5 uDosFileName    = LSA_UNICODE_STRING ptr -8
.text:7C9142F5 DosFileName     = dword ptr  8
.text:7C9142F5 NtFileName      = dword ptr  0Ch
.text:7C9142F5 FilePart        = dword ptr  10h
.text:7C9142F5 Reserved        = dword ptr  14h
.text:7C9142F5
.text:7C9142F5
.text:7C9142F5
.text:7C9142F5                 mov     edi, edi
.text:7C9142F7                 push    ebp
.text:7C9142F8                 mov     ebp, esp
.text:7C9142FA                 push    ecx
.text:7C9142FB                 push    ecx
.text:7C9142FC                 push    esi
.text:7C9142FD                 mov     esi, [ebp+DosFileName]
.text:7C914300                 xor     eax, eax
.text:7C914302                 cmp     esi, eax
.text:7C914304                 jz      short loc_7C914343
.text:7C914306                 push    esi             ; Str
.text:7C914307                 call    _wcslen
.text:7C91430C                 shl     eax, 1
.text:7C91430E                 pop     ecx
.text:7C91430F                 lea     ecx, [eax+2]
.text:7C914312                 cmp     ecx, 65534
.text:7C914318                 jnb     name_too_long
.text:7C91431E                 lea     ecx, [eax+2]
.text:7C914321                 mov
[ebp+uDosFileName.MaximumLength], cx
.text:7C914325
.text:7C914325 loc_7C914325:                           ; CODE XREF:
.text:7C914325                 push    [ebp+Reserved]  ; a4
.text:7C914328                 mov     [ebp+uDosFileName.Length], ax
.text:7C91432C                 push    [ebp+FilePart]  ; a3
.text:7C91432F                 lea     eax, [ebp+uDosFileName]
.text:7C914332                 push    [ebp+NtFileName] ; NtFileName
.text:7C914335                 mov     [ebp+uDosFileName.Buffer], esi
.text:7C914338                 push    eax             ; DosFileName
.text:7C914339                 call
_RtlDosPathNameToNtPathName_Ustr@16 ;
.text:7C91433E
.text:7C91433E loc_7C91433E:                           ; CODE XREF:
```

```
.text:7C91433E                     pop     esi
.text:7C91433F                     leave
.text:7C914340                     retn    10h
.text:7C914343
.text:7C914343 loc_7C914343:                                  ; CODE XREF:
.text:7C914343                     mov
[ebp+uDosFileName.MaximumLength], ax
.text:7C914347                     jmp     short loc_7C914325
.text:7C914347 _RtlDosPathNameToNtPathName_U@16 endp
```

The first important check is detailed in the following subsection.

First check

```
.text:7C9142FD                     mov     esi, [ebp+DosFileName]
.text:7C914300                     xor     eax, eax
.text:7C914302                     cmp     esi, eax
.text:7C914304                     jz      short loc_7C914343
.text:7C914306                     push    esi                ; Str
.text:7C914307                     call    _wcslen
.text:7C91430C                     shl     eax, 1
.text:7C91430E                     pop     ecx
.text:7C91430F                     lea     ecx, [eax+2]
.text:7C914312                     cmp     ecx, 65534
.text:7C914318                     jnb     name_too_long
```

If our path is not shorter than:

65,534 – 2(=UNICODE_NULL) (65532/2 = 32,766 WCHAR symbols)

The function returns false, in any other case the **RtlDosPathNameToNtPathName_Ustr** api function is called.

```
.text:7C914325                     push    [ebp+Reserved]   ; a4
.text:7C914328                     mov     [ebp+uDosFileName.Length], ax
.text:7C91432C                     push    [ebp+FilePart]   ; a3
.text:7C91432F                     lea     eax, [ebp+uDosFileName]
.text:7C914332                     push    [ebp+NtFileName] ; NtFileName
.text:7C914335                     mov     [ebp+uDosFileName.Buffer], esi
.text:7C914338                     push    eax                ; DosFileName
.text:7C914339                     call
_RtlDosPathNameToNtPathName_Ustr@16 ;
```

Since the **RtlDosPathNameToNtPathName_Ustr** API is quite long, we will only paste the most important chunks.

Second check

```
.text:7C9140A2                     mov     eax, [ebp+DosFileName]
.text:7C9140A5                     mov     esi, [ebp+NtFileName]
.text:7C9140A8                     mov     edi, [ebp+a3]
.text:7C9140AB                     mov     [ebp+var_260], edi
.text:7C9140B1                     mov     ebx, [ebp+a4]
.text:7C9140B4                     xor     edx, edx
.text:7C9140B6                     mov     [ebp+pNtFileName], edx
.text:7C9140BC                     mov     [ebp+localFullDosPath], edx
.text:7C9140C2                     mov     [ebp+var_264], 20Ah
```

```
.text:7C9140CC                    mov     ecx, [eax]
.text:7C9140CE                    mov     [ebp+var_240], ecx
.text:7C9140D4                    mov     eax, [eax+4]
.text:7C9140D7                    mov     [ebp+var_23C], eax
.text:7C9140DD                    cmp     cx, 8
.text:7C9140E1                    jbe     short loc_7C9140ED
.text:7C9140E3                    cmp     word ptr [eax], ‘\’
.text:7C9140E7                    jz      loc_7C9182C2

      [...]

.text:7C9182C2                    cmp     word ptr [eax+2], '\'
.text:7C9182C7                    jnz     loc_7C9140ED
.text:7C9182CD                    cmp     word ptr [eax+4], '?'
.text:7C9182D2                    jnz     loc_7C9140ED
.text:7C9182D8                    cmp     word ptr [eax+6], '\'
.text:7C9182DD                    jnz     loc_7C9140ED
.text:7C9182E3                    mov     [ebp+longPathFlag], 1
.text:7C9182EA                    jmp     isLongPath
```

The first check is whether our path is longer than 8 bytes and if so, the routine checks whether our path contains the “long path prefix”. If the “long path prefix” is found our path will be treated as a long path and the **longPathFlag** is set to true:

```
.text:7C9182E3                    mov     [ebp+longPathFlag], 1
```

The flag is in turn checked here:

```
 isLongPath:
.text:7C91412C                    call    sub_7C91040D
.text:7C914131                    mov     [ebp+var_229], 1
.text:7C914138                    and     [ebp+ms_exc.disabled], 0
.text:7C91413C                    mov     [ebp+ms_exc.disabled], 1
.text:7C914143                    cmp     [ebp+longPathFlag], 0
.text:7C91414A                    jnz     handle_longPath
```

We assume that our path contains the **“\\?\”** prefix, this means there is no more checking for us and our parameters will land in the **RtlpWin32NTNameToNtPathName_U** API.

```
.text:7C928F9E handle_longPath
.text:7C928F9E              push    ebx      ; Reserved
.text:7C928F9F              push    edi      ; FilePart
.text:7C928FA0              push    esi      ; NtFileName
.text:7C928FA1              lea     eax, [ebp-240h]
.text:7C928FA7              push    eax      ;DosFileName
.text:7C928FA8              call    RtlpWin32NTNameToNtPathName_U@16 ;
```

Where the DosFileName is converted to NtFileName by replacing “\\?\” with “\??\”.

Having said this, how does the function behave when there is no “\\?\” prefix?

2.1. Path limited to MAX_PATH

If our path does not contain "\\?\" we are going to be limited by the following code sequences:

```
.text:7C9140E7                  jz      loc_7C9182C2
.text:7C9140ED
.text:7C9140ED loc_7C9140ED:
.text:7C9140ED
.text:7C9140ED                  mov     [ebp+longPathFlag], 0
.text:7C9140F4                  lea     eax, [ebp+var_228]
.text:7C9140FA                  mov     [ebp+localFullDosPath], eax
.text:7C914100                  mov     ecx, 538
.text:7C914105                  mov     [ebp+var_264], ecx
.text:7C91410B                  mov     eax, large fs:18h
.text:7C914111                  push    ecx
.text:7C914112                  push    edx
.text:7C914113                  mov     eax, [eax+30h]
.text:7C914116                  push    dword ptr [eax+18h]
.text:7C914119                  call    _RtlAllocateHeap@12 ;
.text:7C91411E                  mov     [ebp+pNtFileName], eax
```

As we can see in the previous assembly snippet, a NT style path is allocated 538 bytes (269 symbols).

Looking further into the code we can notice a call to the **RtlGetFullPathName_Ustr** API with the following parameters:

```
DWORD WINAPI RtlGetFullPathName_U
(
      PUNICODE_STRING DosFileName,
      ULONG size,
      WCHAR* buffer,
      WCHAR** file_part,
      PBOOLEAN invalidName,
      void*  inputPathType
)

.text:7C91415E                  push    edi
.text:7C91415F                  push    [ebp+ localFullDosPath]
.text:7C914165                  mov     edi, 520
.text:7C91416A                  push    edi
.text:7C91416B                  lea     eax, [ebp+var_240]
.text:7C914171                  push    eax ; DosFileName
.text:7C914172                  call    _RtlGetFullPathName_Ustr@24 ;
```

According to the **RtlGetFullPathName_U** parameters the buffer is limited to 520 bytes (260 symbols), is this value familiar? Of course, this value is strictly related to MAX_PATH and we now see where exactly this value is enforced. At a glance, **RtlGetFullPathName_U** is responsible for building the full path given the current format, i.e. if the current path passed to **RtlGetFullPathName_U** is a relative path, it will add the current directory to it:

Some_dir\file.ext → **C:\currentDir\**Some_dir\file.ext

When the path is initially a fullpath it will remain as it is:

C:\foo\bar.ext

The function returns the amount of bytes copied to the buffer if it was successful, but when the buffer is too small it returns the amount of bytes necessary to hold the path. The edge case being 0.

After the call to this API there is a piece of code that checks the returned value:

```
.text:7C914177                   mov     [ebp+var_250], eax
.text:7C91417D                   cmp     [ebp+var_22A], 0  ; invalidName
always set to 0
.text:7C914184                   jnz     problems_with_paths
.text:7C91418A                   test    eax, eax ; error when api
returned 0
.text:7C91418C                   jz      problems_with_paths
.text:7C914192                   cmp     eax, edi ;  necessaryLength >
260 (MAX_PATH)
.text:7C914194                   ja      problems_with_paths
```

As we can see, if the value returned by **RtlGetFullPathName_U** is bigger than **MAX_PATH**, situation that means that the path could not be copied to the buffer, **RtlDosPathNameToNtPathName_Ustr** will return false.

At this point there is only one thing to clarify. A path in NT style is allocated 538 bytes but only 520 bytes are used, this means that 18 bytes must be reserved for something.

This is indeed the case, these 18 bytes are reserved for a different kind of NT prefixes. Some prefixes that I was able to find during my research are as follows:

• _RtlpDosAUXDevice	L"AUX"
• _RtlpDosCONDevice	L"CON"
• _RtlpDosDevicesPrefix _RtlpDosDevicesUncPrefix	L"\??\" L" \??\UNC\"
• _RtlpDosLPTDevice	L"LPT"
• _RtlpDosNULDevice	L"NUL"
• _RtlpDosPRNDevice	L"PRN"
• _RtlpDosSlashCONDevice	L" \\.\CON"

3. Tests

3.1. Antivirus software

We tested the different antivirus products with malware located at very long paths, in order to do so I wrote a small application which creates directory stack with 127 and levels and with a path length of 32,518 symbols.

Here is the code for such application:

```
int main(int argc, char* argv[])
{
      wstring drive = L"\\\\?\\C:";
      wstring dir(255,'Z');
      wstring backslash = L"\\";

      while(1)
      {
            drive.append(backslash).append(dir);
            if( !CreateDirectory(drive.c_str(),0) )
                break;
      }
      CopyFileW(L"catchme.exe",drive.c_str());
      //where catchme.exe is a malicious file
      return 0;
}
```

Some readers will notice that I did not exhaust the allowed path length, the reason behind this is that I assumed that if the developer of the application is not aware of the long path existence, then even a small excess over MAX_PATH will cause trouble. Additionally, we still need some space for the file name.

The result of the tests can be found in the following table:

AntiVirus	Version	Detection	Disinfection	Additional info
Sophos	9.5.1	**NO**	**NO**	*After scan long path AV is not capable to scan anything else, plus win32 application crash.*
Norton	17.6.0.32	YES	YES	*MCUI32.exe can not handle long path and application crash. Buffer Overflow.*
TrendMicro	17.50.0.1366	**NO**	**NO**	
ESET	4.2.42.3	YES	YES	
AVIRA	10.0.0.567	YES	**NO**	*After malware detection in long path when user move mouse cursor on window with details avscan.exe crash.*

AVAST	5.0.594.0	YES	**NO**	
AVG	9.0.0.851	YES	**NO**	
BitDefender	6.00.3790.0	YES	YES	*Report doesn't contain full path to deleted file.*
F-Secure	1.30.15265.0	**NO**	**NO**	
Kaspersky	11.0.0.232	**NO**	**NO**	
Panda	9.01.00	**NO**	**NO**	
*Emsisoft Commandline Scanner	5.0	YES	YES	
*AhnLab - Smart Defense Scanner for Windows Console	2.0.0.11	**NO**	**NO**	
*AVL SDK 2.0 Powered by Antiy Labs	2.0.3.7	**NO**	**NO**	
*Authentium Commandline Scanner	5.2.0	**NO**	**NO**	
*Comodo Antivirus Console Scanner	4.0	**NO**	**NO**	*Buffer Overflow*
*Dr.Web Scanner for Windows	5.00.0.0905070	**NO**	**NO**	
*Fortinet Scanner	4.1.143	**NO**	**NO**	
*FRISK Software International	4.6.1.107	**NO**	**NO**	
*G DATA AntiVirus Command Line	3.0.8260.919	**NO**	**NO**	*Buffer Overflow*
*ConsoleScan	1.0.0.0	YES	YES	

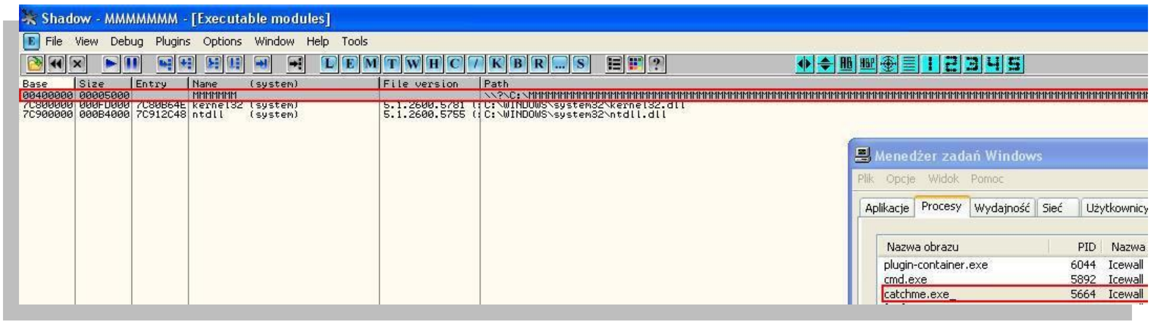
from Jiangmin				
*Microsoft MP Command Line Scanner	1.1.4405.0	YES	**NO**	
*(Norman)NVCC Command Line Scanner	5.99.02	**NO**	**NO**	
*Tachyon Anti-Virus	v2.0,build 1203	**NO**	**NO**	
*PC Tools	7, 0, 3, 5	**NO**	**NO**	*Buffer Overflow*
*Quick Heal	11.00	**NO**	**NO**	
*IKARUS - T3SCAN	1.32.12.0	**NO**	**NO**	
*The Hacker Antivirus	6.5.2.1.356	**NO**	**NO**	
*VirusBlokAda	3.12.14.0	YES	YES	
*Virusbuster Command-line	1.5.6	**NO**	**NO**	
*ViRobot	2008, 8, 6, 131	**NO**	**NO**	

(*) Antivirus engines being used in VirusTotal (www.virustotal.com)
Detection: whether the file was detected as malware.
Disinfection: whether the antivirus was able to disinfect/delete the file.

4. Running processes from a long path

Active processes of executable files located in long paths can cause problems in many applications.

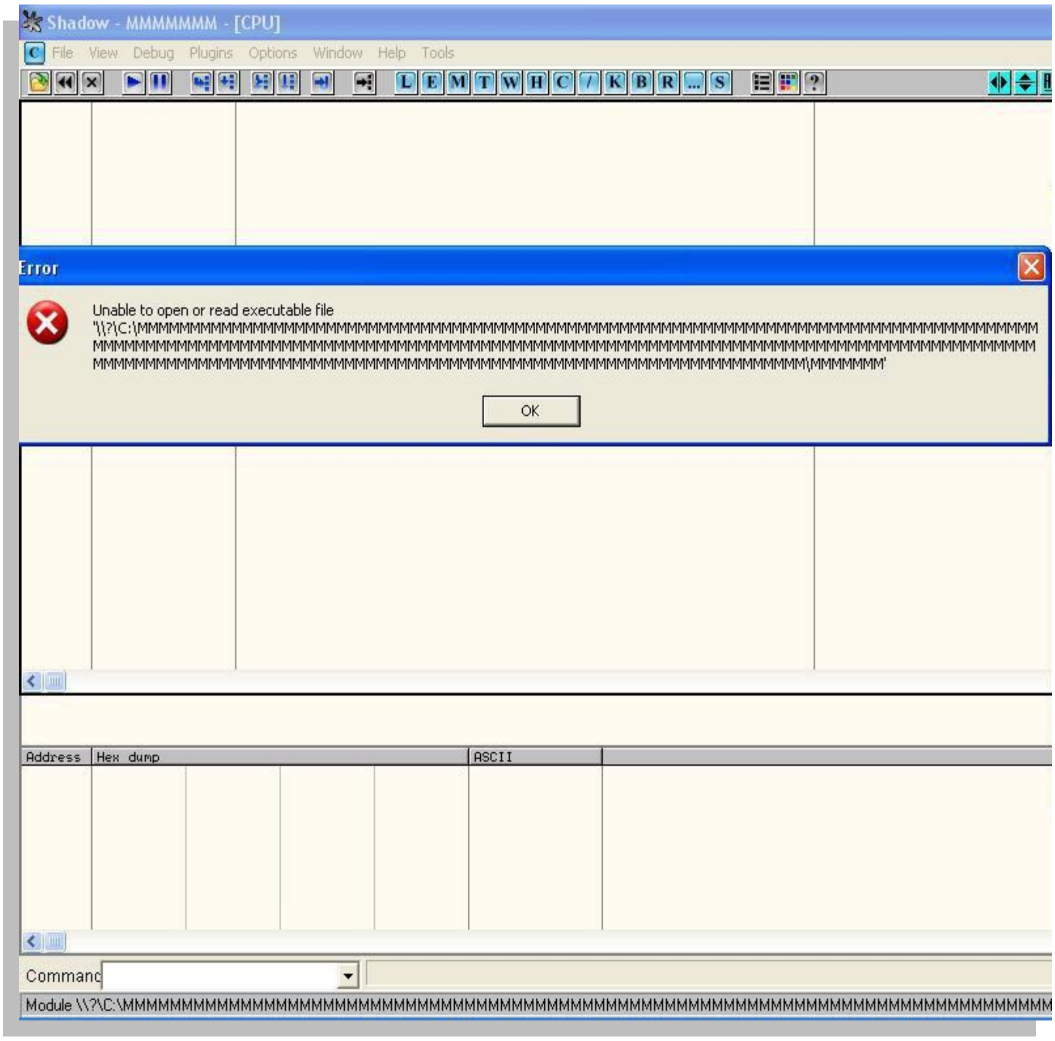
If we try to attach **Ollydbg** to such an application, Ollydbg is not able to load its executable file.



```
Base     Size     Entry    Name      (system)      File version      Path
00400000 00005000          MMMMMMM                                   \\?\C:\MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
7C800000 000FD000 7C80B64E kernel32  (system)      5.1.2600.5781 (:  C:\WINDOWS\system32\kernel32.dll
7C900000 000B4000 7C912C48 ntdll     (system)      5.1.2600.5755 (:  C:\WINDOWS\system32\ntdll.dll
```

```
Nazwa obrazu              PID   Nazwa
plugin-container.exe      6044  Icewall
cmd.exe                   5892  Icewall
catchme.exe_              5664  Icewall
```

Illustration 1 – Attaching Ollydbg to a process with its executable file in a long path



```
Unable to open or read executable file
'\\?\C:\MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM\MMMMMMM'
```

```
Module \\?\C:\MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
```

Illustration 2 - Ollydbg is unable to load the execuutable file

Old versions of **Process Explorer (< v12.04)** and **Process Monitor (< v2.91)** also had problems with long paths, giving rise to buffer overflows.

Problems with SxS

Not every application can be executed from a long path correctly. Some problems appear in CSRSS (more specifically in sxs.dll [SxsGenerateActivationContext]) during an attempt to process data related with side-by-side assembly. Each execution attempt of any application that contains in its resources manifest data ends with the following error code:

ERROR_SXS_CANT_GEN_ACTCTX
14001 (0x36B1)

The application has failed to start because its side-by-side configuration is incorrect.

Please see the application event log or use the command-line sxstrace.exe tools for more detail.

5. Conclusion

Many programmers still use unsecure functions when copying data. Moreover, I would even say that they are even more careless when the data copying is related to paths, probably because developers are used to make use and see paths limited to the “maximal length” of 260 symbols.

The tests performed on the antivirus products have indeed proved this point, this leads me to believe that the tested applications are just a small subset of the software that is affected by long path problem.