Whitepaper

Windows LongPaths: extended-length path

Marcin “Icewall” Noga
martin@hispacec.com
## Index

1. Basic information about paths 3
   1.1. Common lengths of paths and its extended version 3
   1.2. Bypass MAX_PATH limitation with the `\ ??\` prefix 3
2. Path length limitation analysis 3
   FIRST CHECK 5
   SECOND CHECK 5
2.1. Path limited to MAX_PATH 7
3. Tests 8
   3.1. Antivirus software 8
4. Running processes from a long path 11
   PROBLEMS WITH SXS 12
5. Conclusion 13
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
```c
NTSYSAPI
BOOLEAN
RtlDosPathNameToNtPathName_U(
    __in PCWSTR DosFileName,
    __out PUNICODE_STRING NtFileName,
    __out_opt PWSTR *FilePart,
    __reserved PVOID Reserved
);
```

```assembly
; CODE XREF: LdrpMapDll(x,x,x,x,x)+7B8p ...
    public_RtlDosPathNameToNtPathName_U@16 proc near
    _RtlDosPathNameToNtPathName_U@16 proc near
    ; CODE XREF: LdrpMapDll(x,x,x,x,x)+7B8p ...
    uDosFileName = LSA_UNICODE_STRING ptr -8
    DosFileName = dword ptr 0
    NtFileName = dword ptr 0Ch
    FilePart = dword ptr 10h
    Reserved = dword ptr 14h

    mov     edi, edi
    push    ebp
    mov     ebp, esp
    push    ecx
    push    ecx
    push    esi
    mov     esi, [ebp+DosFileName]
    xor     eax, eax
    cmp     esi, eax
    jz     short loc_7C914343
    push    esi
    call    _wcslen
    shl     eax, 1
    push    ecx
    mov     ecx, [eax+2]
    cmp     ecx, 65534
    jnb    name_too_long
    lea     ecx, [eax+2]
    mov     [ebp+DosFileName.MaximumLength], cx
    loc_7C914325:                      ; CODE XREF: LdrpMapDll(x,x,x,x,x)+7B8p ...
    push    [ebp+Reserved] ; a4
    mov     [ebp+DosFileName.Length], ax
    push    [ebp+FilePart] ; a3
    lea     eax, [ebp+DosFileName]
    push    [ebp+NtFileName]; NtFileName
    mov     [ebp+DosFileName.Buffer], esi
    push    eax ; DosFileName
    call    _RtlDosPathNameToNtPathName_Ustr@16 ;
    loc_7C91433E:                      ; CODE XREF:
```

```assembly
```
The first important check is detailed in the following subsection.

**First check**

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.

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

**Second check**
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:

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.

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                loc_7C9140ED:
.text:7C9140ED                mov     [ebp+longPathFlag], 0
.text:7C9140ED                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_Ustr(
    LPCUNICODE_STRING DosFileName,
    ULONG size,
    WCHAR* buffer,
    WCHAR** file_part,
    PBOOLEAN invalidName,
    void* inputPathType
)
```

According to the RtlGetFullPathName_Ustr 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_Ustr is responsible for building the full path given the current format, i.e. if the current path passed to RtlGetFullPathName_Ustr 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:
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` L“\??\”
- `_RtlpDosDevicesUncPrefix` 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:
```c
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:

<table>
<thead>
<tr>
<th>AntiVirus</th>
<th>Version</th>
<th>Detection</th>
<th>Disinfection</th>
<th>Additional info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophos</td>
<td>9.5.1</td>
<td>NO</td>
<td>NO</td>
<td>After scan long path AV is not capable to scan anything else, plus win32 application crash.</td>
</tr>
<tr>
<td>Norton</td>
<td>17.6.0.32</td>
<td>YES</td>
<td>YES</td>
<td>MCGI32.exe can not handle long path and application crash. Buffer Overflow.</td>
</tr>
<tr>
<td>TrendMicro</td>
<td>17.50.0.1366</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>ESET</td>
<td>4.2.42.3</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>AVIRA</td>
<td>10.0.0.567</td>
<td>YES</td>
<td>NO</td>
<td>After malware detection in long path when user move mouse cursor on window with details avscan.exe crash.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Detected</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVAST</td>
<td>5.0.594.0</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>AVG</td>
<td>9.0.0.851</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>BitDefender</td>
<td>6.00.3790.0</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>F-Secure</td>
<td>1.30.15265.0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Kaspersky</td>
<td>11.0.0.232</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Panda</td>
<td>9.01.00</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Emisoft Commandline Scanner</td>
<td>5.0</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>*AhnLab - Smart Defense Scanner for Windows Console</td>
<td>2.0.0.11</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*AVL SDK 2.0 Powered by Antiy Labs</td>
<td>2.0.3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Authentium Commandline Scanner</td>
<td>5.2.0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Comodo Antivirus Console Scanner</td>
<td>4.0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Dr.Web Scanner for Windows</td>
<td>5.00.0.0905070</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Fortinet Scanner</td>
<td>4.1.143</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*FRISK Software International</td>
<td>4.6.1.107</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*G DATA AntiVirus Command Line</td>
<td>3.0.8260.919</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*ConsoleScan</td>
<td>1.0.0.0</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
from Jiangmin

<table>
<thead>
<tr>
<th>Antivirus Engine</th>
<th>Version</th>
<th>Detection</th>
<th>Disinfection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft MP Command Line Scanner</td>
<td>1.1.4405.0</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>*(Norman)NVC C Command Line Scanner</td>
<td>5.99.02</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Tachyon Anti-Virus</td>
<td>v2.0, build 1203</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>PC Tools</td>
<td>7, 0, 3, 5</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*Quick Heal</td>
<td>11.00</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*IKARUS - T3SCAN</td>
<td>1.32.12.0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*The Hacker Antivirus</td>
<td>6.5.2.1.356</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*VirusBlokAda</td>
<td>3.12.14.0</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>*Virusbuster Command-line</td>
<td>1.5.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>*VIRobot</td>
<td>2008, 8, 6, 131</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

(*) 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.

![Illustration 1 – Attaching Ollydbg to a process with its executable file in a long path](image)
Illustration 2 - Ollydbg is unable to load the executable 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.