

May 2005

How Many Bytes in a Zero-Length File?

By Drew Hamre

Here's a riddle that's provoked more than a few nightmares for security administrators: "How many bytes can be stored in what appears to be a zero-length file?" Using Windows *Alternate Data Streams* (ADS), the answer is: "Gigabytes".

ADS is a little-discussed feature of the Windows NTFS file system that surfaced (along with several other NTFS enhancements) in Windows 2000¹. In our day-to-day interactions with files, we perceive each individual file to be a single stream of bytes. Most software we use, including Windows Explorer and the standard operating system I/O routines, treat files in this manner. This is all misleading.

Rather than being a *single* stream of bytes, files in NTFS can be composed of *many* streams. Each file has a default data stream that is unnamed, and most software interacts with only this stream. However, each file can logically enwrap multiple *named* Alternative Data Streams. Assuming the proper I/O APIs are used and the proper ADS names are referenced, user software has full access to these resources.

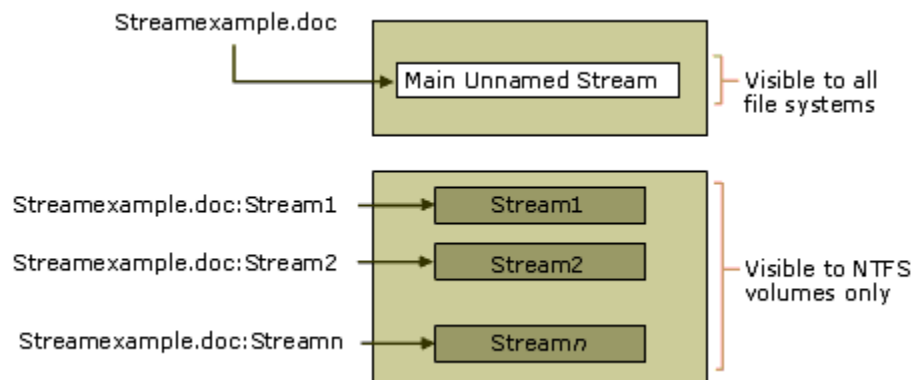
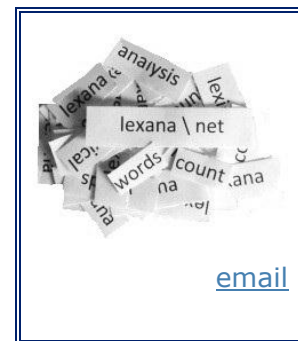


Figure 1 - Microsoft's illustration of the ADS architecture

In this paper we'll demonstrate how to create and manipulate ADSs using both Windows Explorer and common operating system commands. We'll discuss tools for creating, detecting, and editing ADSs. We'll conclude by reviewing the uses of ADS. Some are legitimate. Some are frightening.

Stupid Computer Trick #1: Use ADS to store metadata for .txt files

Most Windows software – including platform utilities such as *Explorer* and the *dir* command – show only the file system’s default data streams. There is no standard Windows utility that either a) shows whether a file has an ADS, or b) displays the size of an ADS if encountered. A file that both *Explorer* and *dir* report as empty may have a hidden ADS that contains gigabytes.

Despite ADS’ absence from the Windows UI, *Explorer*¹ may *covertly* create an ADS for a file. In the steps below, we’ll use *Explorer* to supply Word-like document properties (*Author* and *Version*) to a simple ASCII text file. As we’ll see, *Explorer* will happily report the file is empty, both before *and after* the metadata has been added.

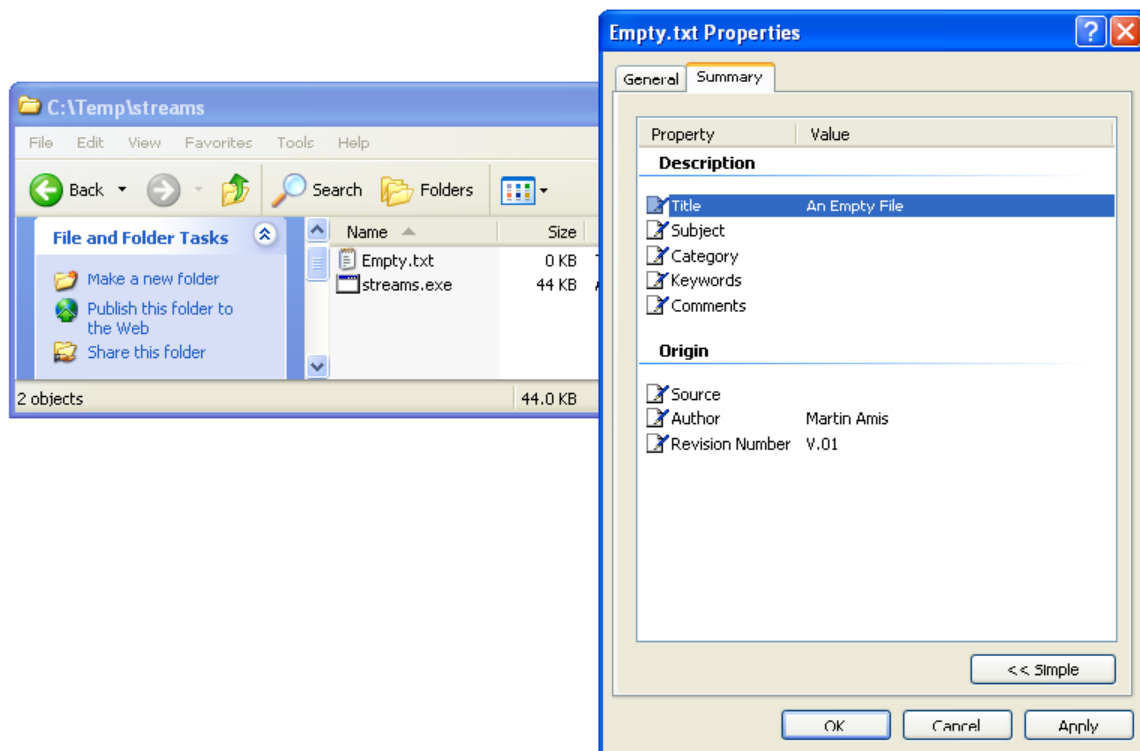


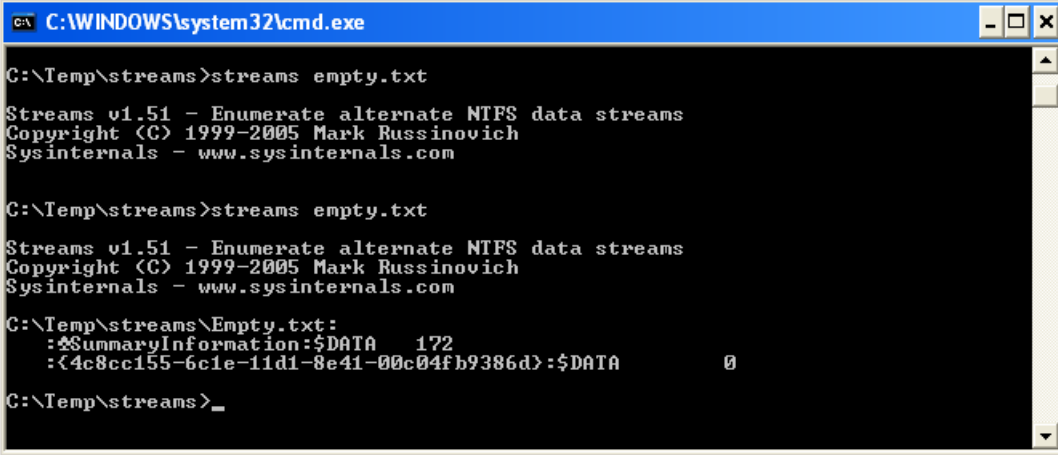
Figure 2 – Explorer loads ADS metadata while file remains “0-bytes” long

To recreate this experiment, use *Explorer* to navigate to a test directory on your system and then: a) create a new empty file (right-click an Explorer pane and click ‘New ...’ | ‘Text Document’); b) verify that Explorer reports the size as “0” bytes; c) display the ‘Summary’ tab of the file’s property sheet (right-click, ‘Properties’), d) enter some values for the pre-configured attributes; and e) accept the property changes and then re-verify that the file size remains 0-bytes.

How do we know the metadata was loaded into the file’s ADS, rather than being stored within the internal directory structures used by NTFS? To resolve this question, we’ll use a third-party ADS detection utility from [SysInternals](#) called *streams.exe*.

¹ Screenshots are from an Administrative account under Windows XP Pro (SP2_gdr.050301-1519).

The *Streams* utility (for which SysInternals provides C source code) scans a file or directory tree for the presence of ADS streams. A sample scan follows:



```
C:\WINDOWS\system32\cmd.exe

C:\Temp\streams>streams empty.txt

Streams v1.51 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2005 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Temp\streams>streams empty.txt

Streams v1.51 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2005 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Temp\streams>streams Empty.txt:
:SummaryInformation:$DATA      172
:<4c8cc155-6c1e-11d1-8e41-00c04fb9386d>:$DATA      0

C:\Temp\streams>_
```

Figure 3 - The 'Streams' utility reports the existence of ADS streams

In the above, *streams* is executed twice: the first scan finds no alternate data streams, whereas the second scan (taken after we added metadata via Explorer) reports two streams: one labeled `:char(5)SummaryInformation`, the second labeled `'4c8cc155 ...'`. Our metadata has been stored in 'SummaryInformation', which contains 172-bytes of textual and binary data.

Throughout our manipulations, the file (*empty.txt*) behaves normally, and it will continue to do so if we add content, rename the file, and so on. All standard platform utilities (Notepad, Explorer, etc.) interact with the default data stream, and give no explicit hints that there are additional streams within the file.

Stupid Computer Trick #2: Add custom properties to a .JPG image

Let's go a step farther and use ADSs to store custom metadata – an XML string holding arbitrary name/value properties that support licensing and collaborative editing. We'll add these extensions to a .JPG image using basic DOS commands.

```

C:\WINDOWS\system32\cmd.exe

C:\>cd temp\streams
C:\Temp\streams>streams plane.jpg

Streams v1.51 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2005 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Temp\streams>type plane_authorship.txt
<Author><Source>Susan Martin</Source><Copyright>SP Times 2005</Copyright><LastEditor></LastEditor></Author>
C:\Temp\streams>type plane_authorship.txt > plane.jpg:Authorship

C:\Temp\streams>streams plane.jpg

Streams v1.51 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2005 Mark Russinovich
Sysinternals - www.sysinternals.com

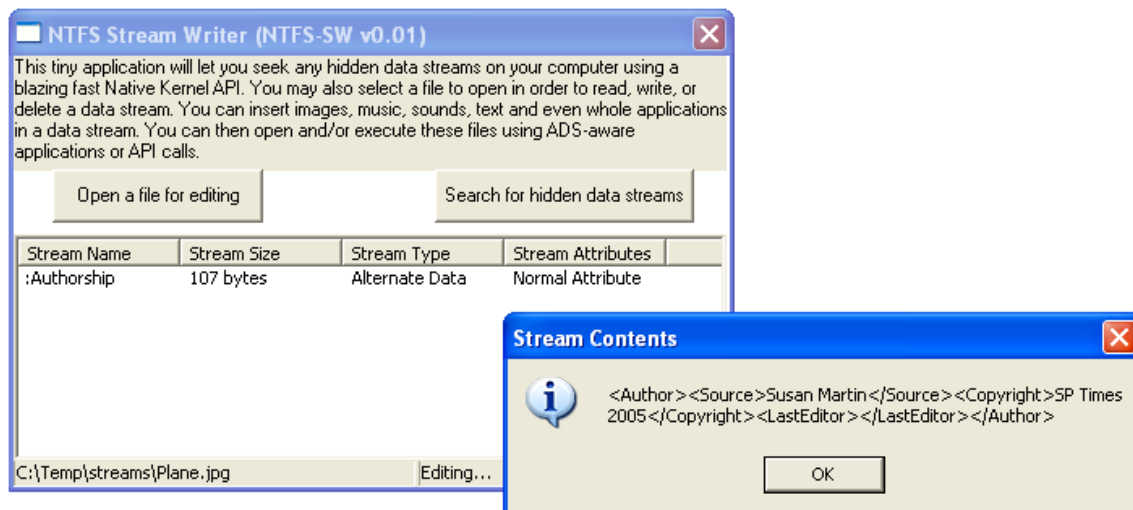
C:\Temp\streams\Plane.jpg:
:Authorship:$DATA 107

C:\Temp\streams>

```

To replicate the above experiment, a) copy an image to a test directory (and make note of its size); b) prepare a text file that contains your metadata string (our sample is listed in the window above); c) and redirect the contents of your metadata file into a new ADS associated with the image file: (e.g., *type metadata.txt > image.jpg:streamname*).

With ADS-detection software (such as SysInternals 'streams') we can verify the existence of the new stream. With an ADS viewer (such as Alex Ionescu's [StreamWriter.exe](#)) we can view the contents of our stream.



As before, the ADS is 'invisible' to most standard platform software. The modified image file will render normally in our default .JPG viewer, and Windows Explorer will report no change in the file's size.

ADS visibility is affected by factors including stream name. In the examples above, we didn't give the ADS name an extension, and if we try to view the stream with Notepad (e.g. *Notepad Plane.jpg:Authorship*) we'll receive a file-not-found dialog.

However, had we applied a .txt extension to the ADS name, we'd be able to read the ADS with Notepad (e.g., *Notepad Plane.jpg:Authorship.txt*).

Windows Indexing Service will create ADSs for image files and load thumbnails

In the examples above, we've loaded textual data into ADSs. Of course, the streams can also hold binary data – a capability exploited by the Windows Indexing Service during image processing. By default, the [Indexing Service](#) will create thumbnails of image files and load these thumbnails into an ADS labeled '?Q30lsl dxJoudresxAAAqpcawXc'. The binary data loaded to an ADS can also include executable code, as we'll now demonstrate.

Stupid Computer Trick #3: Use ADS to hide an EXE

Immediately below, we'll hide an .exe within another .exe (though note it would have been equally straightforward to hide an .exe within a more innocuous file type, such as a .txt or .bmp file).

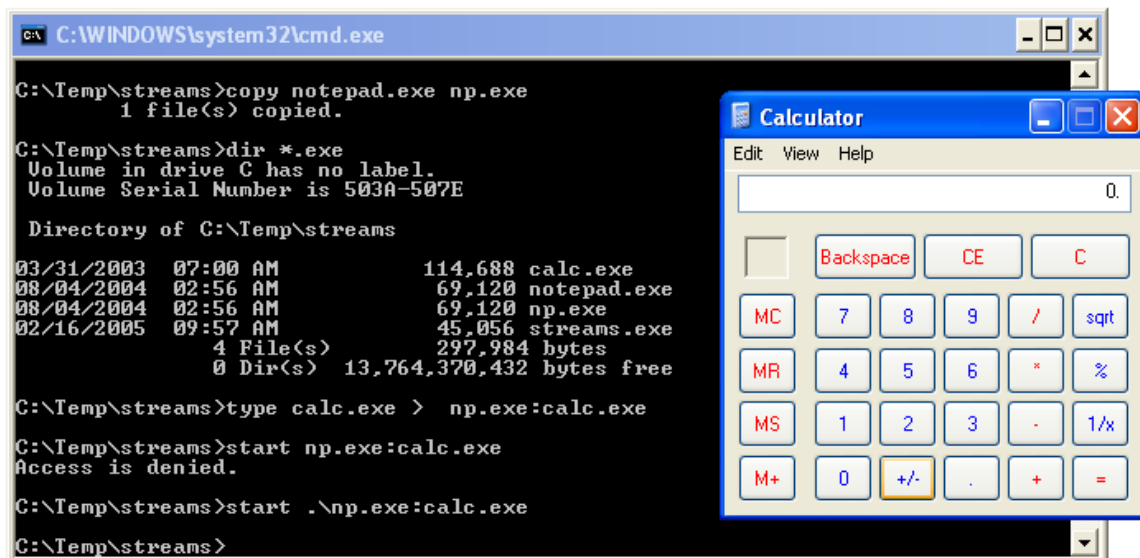


Figure 4 - Hiding and launching calc.exe from within Notepad

To reproduce the above experiment, copy two .exes to a test directory. Use the 'type' command² to copy one of the executable byte streams into an ADS of the second executable. When you invoke the named ADS, note that path information must be provided as in the example above.

Writing ADS-aware software

Microsoft gives a [simple example](#) of how to create an ADS (excerpt in C, below):

```
hStream = CreateFile( "testfile:stream",  
GENERIC_WRITE,  
FILE_SHARE_WRITE,
```

² The 'type' command streams bytes from *point a* to *point b*, whether ASCII text or binary. Alternatively, using the 'copy' command will create the ADS, but the resulting stream won't be executable.

```
NULL,  
OPEN_ALWAYS,  
0,  
NULL );
```

However, programmatically *detecting* ADSs is far more difficult than simple I/O. Sample detection code (too lengthy to include here) can be found in the C-source for SysInternals' *streams* utility and in Visual Basic source for *StreamWriter* (above). These code examples all pre-date .NET, and are basically thin wrappers around the relevant operating system file and I/O APIs.

Calling these APIs from a language like VB is laborious, due to the lack of pre-defined libraries for the requisite data structures. *StreamWriter* is noteworthy as it defines these structures, and is especially laudable for implementing two alternative approaches to ADS detection, one based on the *NtQueryInformationFile()* API, the other based on *BackupRead()*.

Refer to the 'Resources' section (below) for the links to these and other examples.

Worms and other malware exploits of ADS

ADSs first came to widespread attention with the [IIS '\\$DATA' bug](#), which potentially exposed .asp *scripts* (rather than the HTML they emitted) to prying eyes, including connection strings, embedded passwords, and other unfortunate details.

Since then, exploits of ADS include "VBS/Potok-A", a worm that used Outlook to spread itself as an attachment. The worm hid part of itself in an ADS associated with the ODBC.INI file.

Those of us whose job focus isn't security may be taken aback to learn how sophisticated the techniques for hiding malware have become. It's been widely reported that most commercial anti-spyware detectors miss roughly half of all infections. The difficulties in finding hidden software can be inferred from the pain caused by the solutions posing as cures.

The latest hidden-software detector from Microsoft Research ([Strider Ghostbuster](#)) is essentially a brute force solution with the following steps: a) create a comprehensive directory listing using the potentially infected OS; b) create a comprehensive directory listing using a clean read-only copy of Win/PE; and c) compare the two listings via a read-only copy of *WinDiff*.

Even this laborious process is limited to finding spyware that's hidden by rootkits (e.g., hacker toolkits for stealth operations such as subverting the *FindFirstFile / FindNextFile* APIs). *Strider* won't find malware "that hides in BIOS, Video card EEPROM, disk bad sectors, *Alternate Data Streams*, etc" (emphasis added). For these reasons, most computer 'forensic' toolkits include utilities to detect files with ADSs, and Frank Heyne's freeware tool *LADS* (below) is a popular choice.

Beyond security, ADS also raises concerns over [disk space management](#). Just as *Explorer* and *dir* don't show space allocated to ADSs, neither do they include ADS allocations in free space calculations. In other words, I may think I have 2GB free, but if there's a 2GB ADS lurking on my drive, I could run out of space at the next keystroke. As a palliative, Microsoft notes that *chkdsk* "accurately reports space used by a user's data files, including alternate data streams."

FAT Kills: Removing ADSs from a file

Because ADS requires NTFS, the simplest process for removing an unwanted ADS from a file is to *copy* it to a non-NTFS volume. (*Copy* will issue a data loss warning in these circumstances, and you can decide whether to cancel or proceed.) In typical use, ADSs may be harder to preserve than remove because common file transfer practices (FAT-formatted USB stick, email) will generally strip ADSs from the file.

The many legitimate uses of ADS

If you've loaded software that *requires* an NTFS partition, that software probably requires ADS. Exchange Server 2003, for example, uses ADS to manage states of message transport under SMTP. Other software uses ADS to hold 'undo' information. Some anti-virus programs write checksums into each scanned file's ADS. The Win2K indexing service (as we've seen) stashes thumbnails in each graphic file's ADS. As we've also seen, *Explorer* allows certain pre-defined propertiesⁱⁱ to be stored in a file's ADS, and this could be extended to support arbitrary document properties via a custom ADS handler.

For a variety of reasons, ADS is becoming indispensable to the Windows environment. However, it would better serve Windows security if future generations of the user interface exposed ADS resources and behavior. A simple "Display ADS" option in the next release of *Explorer* would do much to ease concerns.

Summary

This paper demonstrated how to manipulate Alternate Data Streams using standard platform utilities, and discussed available tools for detecting and editing these streams. The paper reviewed several uses of ADS, both harmful and benign.

Resources

How To Use NTFS Alternate Data Streams

<http://support.microsoft.com/default.aspx?scid=kb;en-us;105763>

SysInternals "Streams" utility and source

<http://www.sysinternals.com/ntw2k/source/misc.shtml#streams>

NTFS Data Streams: the true way to hide information and extend your file system

by Alex Ionescu

<http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=47299&lngWId=1>

LADS - List Alternate Data Streams by Frank Heyne

http://www.heysoft.de/Frames/f_sw_la_en.htm

Digital village: Wading into alternate data streams by Hal Berghel & Natasa Brajkovska

[Communications of the ACM Volume 47, Number 4 \(2004\), Pages 21-27](#)

The Dark Side of NTFS (Microsoft's Scarlet Letter) by H. Carvey

<http://www.infosecwriters.com/texts.php?op=display&id=53>

Author Filling or Changing Custom Properties

<http://www.codecomments.com/archive299-2004-8-268452.html>

Acknowledgement

Thanks to reviewers for their time and suggestions; any inaccuracies remain mine.

[Drew Hamre](#) is a principal of [lexana\|net](#) and lives in Golden Valley, Minnesota.

ⁱ Microsoft's Windows 2000 included substantial enhancements to NTFS, including encryption, disk quotas, reparse points, volume mount points, sparse files, distributed link tracking, and (accessible) alternate data streams. It's generally held that ADS resulted from Microsoft's need to interoperate with the Apple Macintosh's *Hierarchical File System*, and its constructs of a file's *resource* and *data fork*.

ⁱⁱ The ability to store extended, application-specific file properties in an ADS is one of its most enticing uses, especially where the application file format itself doesn't support such extensions. Note the familiar extended properties of Microsoft Office documents aren't stored in ADS, but rather are stored in the default stream as part of Office's [OLE 2 Compound Document](#) format.