

Wading into Alternate Data Streams

The open-ended nature of ADSs makes them an extremely powerful Windows resource worthy of deeper exploration.

The concept of non-monolithic file systems is not new. “File forks” were an integral part of the original Macintosh Hierarchical File System. Apple separated file information into data and resource forks, where the resource fork contains a resource map followed by the resources, or links to resources, needed to use the data. Typical resources would include program code, icons, font specifications, last location of application window, and other file-related metadata that is not stored in the disk catalog (for example, file signatures). File forks may be null, and many files contain both forks, the accidental separation of which produces the dreaded “error 39.” The Macintosh Resource Manager could control up to 2,700 resources per file, though the linear search, single-access control strategy makes managing more than a few hundred resources cumbersome and slow. File forks remained in use until Apple introduced OS X. Built on a Linux kernel, OS X stores resources in separate .src files.

Microsoft introduced non-monolithic file structures in its New Technology File System (NTFS) along with the Windows NT operating system. Since the loss of a non-null resource fork renders the data fork useless, Microsoft had to accommodate non-monolithic files for compatibility with

Macintosh files and Apple Talk. Among the many innovations in NT, Microsoft ported non-monolithic file structures from the Macintosh world over to the PC. In Microsoft parlance, non-monolithic file structures are called file streams.

Alternate Data Streams

One may make reasonable comparisons between the Macintosh data and resource forks and the Microsoft primary and alternate data streams, respectively. In NTFS the primary data stream (aka default data stream or unnamed data stream or, simply, file) is called \$DATA. A large number of alternate data streams (ADSs) may be associated with a single primary data stream (PDS). We emphasize that ADSs are associated with, and not attached to, primary data streams. The associations are maintained in the Master File Table (MFT), and managed by a variety of application program interfaces (APIs). As a simple illustration, a right mouse click on any NTFS file and subsequent selection of the properties tab will recover ADS metadata through a standard Windows MFT API.

Microsoft’s approach to non-monotonic file structures has some unusual properties:

- Anything digital can become an ADS: thumbnails

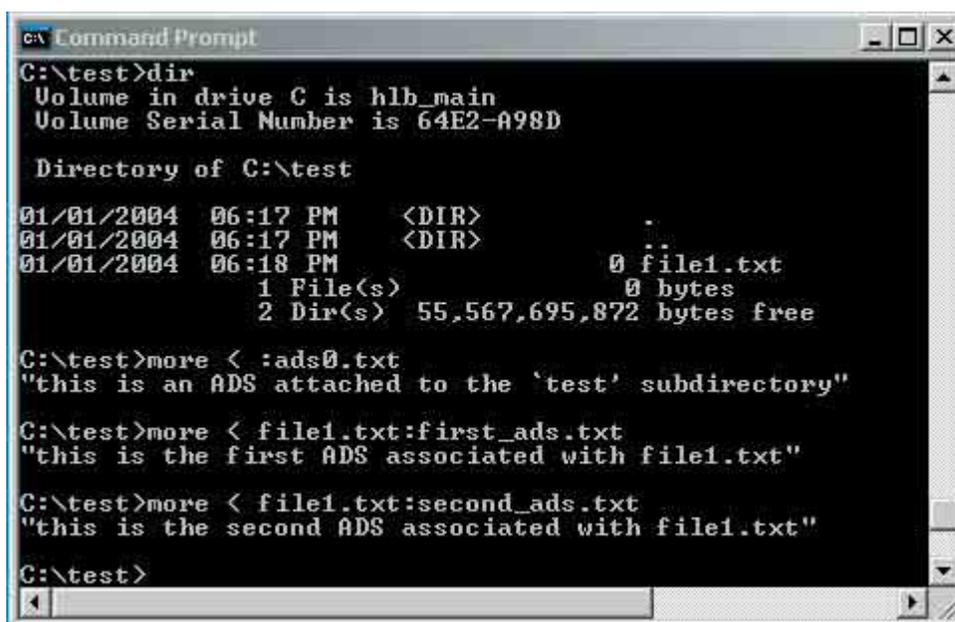
and icons, metadata, passwords, permissions, encryption hashes, checksums, links, movies, sound files, binaries, whatever. We'll return to the "whatever" in a moment.

- NTFS supports a very large number of ADSs. The Purdue COAST project estimated an upper bound of 6,748 ADSs per primary data stream on Win-

seems to us to be an unnecessarily arbitrary and shortsighted limitation.

The best way to get a feel for ADS is with some hands-on experience. Our example shown in the sidebar here may seem clumsier than necessary at first, but it has the benefit of simplicity because the

entire demonstration can be completed within a single DOS command prompt window.



```
c:\ Command Prompt
C:\>dir
Volume in drive C is hlb_main
Volume Serial Number is 64E2-A98D

Directory of C:\test

01/01/2004  06:17 PM    <DIR>          .
01/01/2004  06:17 PM    <DIR>          ..
01/01/2004  06:18 PM                0 file1.txt
                1 File(s)          0 bytes
                2 Dir(s)    55,567,695,872 bytes free

C:\>more < :ads0.txt
"this is an ADS attached to the 'test' subdirectory"

C:\>more < file1.txt:first_ads.txt
"this is the first ADS associated with file1.txt"

C:\>more < file1.txt:second_ads.txt
"this is the second ADS associated with file1.txt"

C:\>
```

Figure 1. Recovering hidden Alternate Data Streams.

Phishing and Executable Streams

As previously stated, ADSs may contain anything—text, images, sound and video files—anything. The most interesting type of “anything” is the binary executable. Presuming readers of this column have completed the example in our sidebar and are up for the challenge, we'll

now attach an executable to an empty text file. In this case we're assuming the Windows XP system calculator is stored in the location C:\windows\system32\calc.exe. If not, create a path to the harmless executable of choice.

We now rename <calc.exe> as the ADS, <d.exe>, and associate it with the empty text file <test.txt>:

- Since ADSs are separate files, they appear hidden from applications that call routine Windows APIs. The use of ADS is completely transparent to standard Windows file management tools—it's as if they don't exist. This includes such key system applications as Windows Explorer and Task Manager, and core command line executables like DIR.

- The normal data stream ADS API syntax in the Windows NT/2000/XP series is <filename>:<ADSname>:<ADStype> (where ADStype is optional).
- ADSs form a tree structure that has a maximum depth of 1. The decision to prevent nesting of ADSs

now attach an executable to an empty text file. In this case we're assuming the Windows XP system calculator is stored in the location C:\windows\system32\calc.exe. If not, create a path to the harmless executable of choice.

We now rename <calc.exe> as the ADS, <d.exe>, and associate it with the empty text file <test.txt>:

```
C:\...\test>type c:\windows\system32\calc.exe > .\test.txt:d.exe
```

and execute the ADS directly:

```
C:\...\test>start .\test.txt:d.exe
```

You should see the Windows calculator appear on

ADS Example

This exercise was run on Windows XP Pro with administrative privileges. Similar results would result if it were run under Windows 2000 or even Windows NT, with slight adjustments to the syntax.

First, open a DOS command prompt window. Then make and move into a new directory—in our case <test>. This step is important, because we'll need to remove this entire directory after our experiment to restore the computer to its pre-experiment state. Do not perform this experiment in a root directory, or any other directory that you are unwilling to erase. The commands

```
mkdir test
cd test
```

will create a new directory, <test>, to experiment in. We then create the simplest of ADS, one that is attached to a folder, as follows:

```
C:\...\test>echo "this is an ADS attached
to the 'test' subdirectory" > :ads0.txt
```

A display of the contents of the directory:

```
C:\...\test>dir
```

reveals an empty directory. How can that be? This is where the NTFS sleight-of-hand comes in: ADS are hidden to Windows applications that rely on the standard MFT APIs. "DIR" is one such example, as is Windows Explorer and the file properties box.

In the preceding example, since the PDSname field is null, <ads0.txt> is by default associated with the subdirectory name in the MFT. Directories in Windows are themselves files that reference other files.

Next, we'll attach an ADS to an empty file:

```
C:\...\test>echo "this is the first ADS
associated with file1.txt">
file1.txt:first_ads.txt
```

Note that we never created <file1.txt> to begin with. Since the colon is not a legitimate filename character, the Windows command processor understands

that the data to be echoed is intended for the associated ADS. Since the ADS has to be associated with something in the MFT, Windows conveniently creates an empty file named <file1.txt> for the association. In Microsoft parlance, we think of this as a single file where <file1.txt> is the primary/default/unnamed data stream, and a named stream labeled <first_ads.txt>.

This time the display of the contents of the directory reveals only an empty file <file1.txt>. Now, let's add a second ADS to file ads1.txt

```
C:\...\test>echo "this is the second ADS
associated with file1.txt">
file1.txt:second_ads.txt
```

Repeat

```
C:\...\test>dir
```

and observe that nothing has changed. We still have an empty file, <file1.txt>, in a directory consisting of 0 bytes.

Appearances to the contrary, we may confirm that these ADSs do indeed exist by typing the following:

```
C:\...\test>more < :ads0.txt
C:\...\test>more <file1.txt:first_ads.txt
C:\...\test>more <file1.txt:second_ads.txt
```

Your results should look like those in Figure 1.

Of course, the more useful ADS will be associated with non-null files. To wit:

```
C:\...\test>echo "this is the data for
file 2" > file2.txt
C:\...\test>echo "this is the data for
ADS1 of file 2" > file2.txt:ads1.txt
C:\...\test>echo "this is the data for
ADS2 of file 2" > file2.txt:ads2.txt
```

The remaining 6,746 repetitions are left to the reader. **C**

your screen. A directory listing still reveals only primary data streams. The executable is entirely hidden, but very much there—and obviously active—as can be confirmed by looking at the active processes listing under Windows Task Manager by pressing the <CTL><ALT> keys simultaneously (see Figure 2).

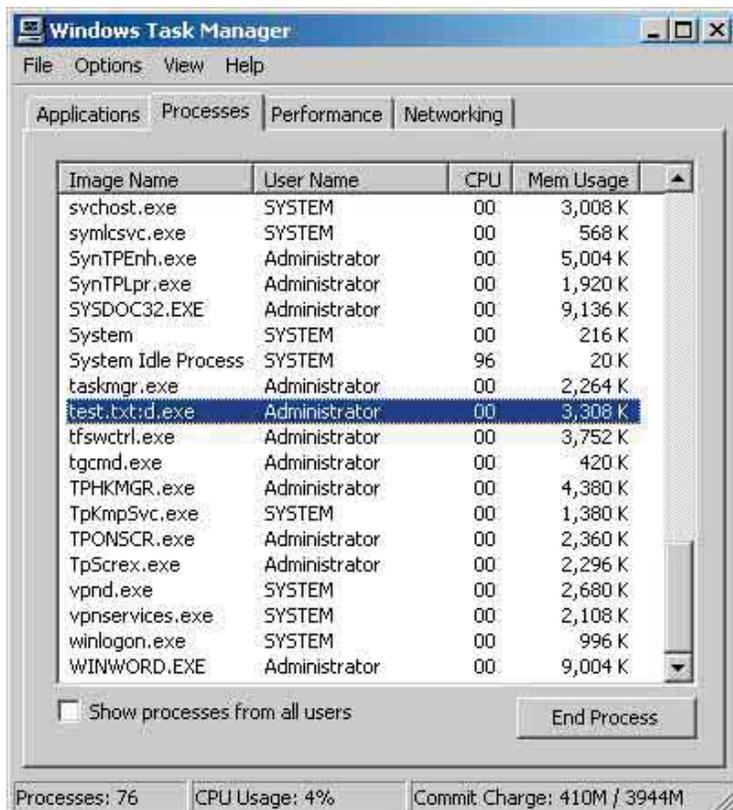


Figure 2. Windows Task Manager's report of the Windows calculator executing as the Alternate Data Stream <test.txt:d.exe>.

Of course, masking an executable by just changing the filename is neither clever nor particularly deceptive, so a hacker might add the requisite stealth by invoking the native Windows Scripting Host with the control parameters set to execute files with non-executable file extents. In this way one could rename <trojan.exe> as something innocuous like <help.fil>,

It is apparent that with minimal effort one can sufficiently mask the hidden executable so that its function is obscured. Of

and execute it with WSH.

It is interesting to note that prior to Windows XP, the ADS didn't even appear in the process listing. Had we hidden the ADS behind something innocuous like <cmd.exe> or <notepad.exe>, the execution of the ADS would be undetected.

The hiding of the function behind an innocuous appearing executable is akin to Internet scams where the unsuspecting are lured to spoofed Web sites that appear harmless while actually harvesting personal or private information—a technique called “phishing.” For lack of a better term, we may think of planting hostile executables in ADS as “file phishing”: creating an environment in which things are not as they appear.

Before we proceed, let's clean up the data streams, directories, and files on your computer. ADSs can only be deleted if their associated primary data stream is deleted, so once you're done experimenting, erase all of the files in your test directory, go up one directory and remove the directory itself, or simply erase the entire <test> directory with Windows Explorer.

At this point you're back where you started, no worse for wear.

NTFS Master File Tables

To understand ADS, one must investigate the way the Windows MFT record works. The MFT is a relational database in which the records correspond to files and the columns to file attributes. Following 16 records of metadata files, there is at least one MFT record for each file and folder (subdirectory) on all hosted disk volumes. All records are 1Kb in size, allowing the data and attributes of very small files or folders to be stored in an MFT record itself. Larger files and folders are referenced by pointers within their records. Folders are externally organized as B-trees.

Each record contains basic file attributes relating to date and time stamps, permissions, filename and extent, security descriptors, and so forth. The utility

of the MFT results from the fact that it is set up to automatically use external pointers for all data that cannot fit within the 1Kb record. Having this in place allows virtually unrestricted, scalable control over file management. All data streams (regardless of whether they're primary or alternate) maintain all of the necessary information for the Windows APIs to manipulate them: for example, allocation size, actual data length, whether they're compressed or encrypted, and so forth. Given this situation, the proliferation of data streams involves little more than the proliferation of pointers within the MFT. The only mitigating factor is that the ADSs cannot be nested, which means any ADS file organization beyond a one-level deep tree would have to be accomplished at the applications layer.

As we indicated, the low-level Windows APIs (such as CreateFile, DeleteFile, ReadFile, WriteFile) were designed to treat all data streams the same, ADSs or PDSs. Under NTFS, ADS support is completely transparent at that level. The weakness is that the higher-level utilities (DIR, Windows Explorer, Task Manager) were not intended to support ADS. This is where the truly baroque nature of Microsoft's ADS design logic makes itself known. One can use the low-level APIs to manipulate ADSs easily, but the higher-level utilities conceal their presence. From the end user's perspective, it's hard to delete a data stream without first being able to find it! Fortunately there are third-party utilities such as Lads, ScanADS, Streams, and Crucial that help locate ADS by working directly with the low-level APIs (especially, the "backup_" functions). Figure 3 illustrates their use on our <test> directory after we completed the experiment described previously. Note that Streams requires a separate test for folder ADS (remove the "-s" parameter). Crucial has a GUI interface and only scans entire drives, and will not be shown here.

Security Implications of ADSs

A Google search for the phrase "alternate data streams" will yield several thousand hits, most of an alarmist nature. This is unfortunate in many ways, because the power of ADSs has yet to be realized. While it is true that there is malware that takes advantage of ADSs (W2k.stream is one example), that malware has not proven to be as destructive as the more mainstream varieties that rely on buffer

```

C:\test2>scanads -s c:\test
ScanADS 1.1 - Scan Alternate Data Streams
Copyright (C) 2003 Tiago Halm
KodeIT - http://www.kodeit.com

Verbose:      no
Recursive:    yes
Get Data:     no
Server:       localhost
Path:         c:\test

<2 ADS>: c:\test\file1.txt
         (52 bytes) :first_ads.txt
         (53 bytes) :second_ads.txt

<2 ADS>: c:\test\file2.txt
         (40 bytes) :ads1.txt
         (40 bytes) :ads2.txt

<1 ADS>: c:\test
         (55 bytes) :ads0.txt

C:\test2>streams -s c:\test
Streams v1.5 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2003 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\test\file1.txt:
  :first_ads.txt:$DATA 52
  :second_ads.txt:$DATA 53
c:\test\file2.txt:
  :ads1.txt:$DATA 40
  :ads2.txt:$DATA 40

C:\test2>lads /S c:\test
LADS - Firmware version 3.21
(C) Copyright 1998-2003 Frank Heyne Software (http://www.heysoft.de)
This program lists files with alternate data streams (ADS)
Use LADS on your own risk!

Scanning directory c:\test\ with subdirectories

   size  ADS in file
-----
   55    c:\test\ads0.txt
   52    c:\test\file1.txt:first_ads.txt
   53    c:\test\file1.txt:second_ads.txt
   40    c:\test\file2.txt:ads1.txt
   40    c:\test\file2.txt:ads2.txt

```

Figure 3. Typical ADS reporting utilities at work.

overflows, NetBIOS and RPC vulnerabilities, session hijacking, or address spoofing. As a datapoint, all W2k.stream threat vectors were assessed "low" by Semantec (see www.sarc.com/avcenter/venc/data/w2k.stream.html).

What created most of the alarm was the "hidden"

nature of ADS combined with the absence of Microsoft utilities that supported direct access and control within native file utilities—but, then, that wasn't why Microsoft built ADS into NTFS in the first place. The mere mention of a hidden feature to

anyone with even a slight anti-Microsoft bias is guaranteed to produce an animated response. Unfortunately, Microsoft added fuel to the fire by failing to include a “display ADS” checkbox as a Windows Explorer configuration option and direct ADS control at the utility level. Most users wouldn't be bothered with ADS management, but full file disclosure would have been comforting to those prone to anxiety attacks.

The facts are less alarming than the several thousand Google hits might lead us to believe. While it is true that ADS could be used as a conduit for malware executables, so can email attachments. Further, modern anti-virus scanners routinely scan for malware in all Windows data streams, including ADS, so the risk of intrusion should be no greater with ADS than PDS.

The same is true for covert channeling. ADS could be used for that purpose, but so could the options field in a normal ICMP packet. With the ease that malware such as Loki conducts encrypted covert data channeling at the IP level, why would a hacker become involved with the applications layer?

The claim that ADSs are difficult to delete is equally misleading. ADS file space gets reallocated in just the same way that PDS and directory space does. File-wiping products such as Cyberscrub (www.cyberscrub.com) even include ADS “scramblers” for extra safety.

By any reasonable measure, ADS vulnerability has been overstated.

Conclusion

Alternate Data Streams have demonstrated considerable potential in object-oriented OSs and application environments, or those that involve complex file and data structures. While Microsoft is officially committed only to the Object Linking and Embedding (OLE) 2.0 model of structured storage, ADS will likely remain with us as long as Windows OSs continue to use NTFS. To quote Microsoft:

“Alternate data streams are strictly a feature of the NTFS file system and may not be supported in future file systems. However, NTFS will be supported in future versions of Windows NT. [including Windows 2000 and XP] Future file systems will support a model

URL Pearls

A good overview of Macintosh file forks is available at the Apple Web site: developer.apple.com/documentation/mac/Files/Files-14.html.

Microsoft's MSDN Library (msdn.microsoft.com/library/) is a reliable source of information on Windows NT operating systems, including APIs and ADS. Detailed documentation on NTFS is available at the Microsoft Technet site (www.microsoft.com/technet/).

The 1998 overview of NTFS by Richter and Cabrera in the *Microsoft Systems Journal* at www.microsoft.com/msj/1198/ntfs/ntfs.aspx is still worth reading. This article also has links to sample code.

The general-purpose Windows utility that is ideal for ADS manipulation is `<cp.exe>`, variations of which may be found in either the Windows Resource Kit or from the Sourceforge.net GNU utilities library at unxutils.sourceforge.net, the latter being free.

Two good sources for the fundamentals of ADS programming are Dino Esposito's *A Programmer's Perspective on NTFS 2000 Part 1: Stream and Hard Link* at msdn.microsoft.com/library/en-us/dnfiles/html/ntfs5.asp?frame=true, and Eugene Kaspersky and Denis Zenkin's, “NTFS Alternate Data Streams,” Document 19878 in *Windows and .NET Magazine* (www.winnetmag.com). Esposito's article contains an interesting code fragment that adds a new property page with stream information. The Code Project (www.codeproject.com/csharp/CsADSDetectorArticle.asp) also contains useful code fragments.

Effective ADS detectors include ScanADS (www.securiteam.com/tools/5JP0C2KAAQ.html), LADS (www.heysoft.de/Frames/f_sw_la_en.htm), CrucialADS (www.crucialsecurity.com), and Streams (www.sysinternals.com/ntw2k/source/misc.shtml).

based on OLE 2.0 structured storage (IStream and IStorage). By using OLE 2.0, an application can support multiple streams on any file system and all supported operating systems (Windows, Macintosh, Windows NT, and Win32s), not just Windows NT.” (See the Microsoft Knowledge Base article “HOWTO: Use NTFS Alternate Data Streams” (number 105763), available at support.microsoft.com/default.aspx?scid=kb;en-us;105763.)

There is no question that ADSs are underutilized in Windows. Like previous major software houses, Microsoft felt compelled to opt in favor of backward compatibility. For example, to use <desktop.ini> files to parse the contents of a directory and <.tmp> files to hold transitory data, seems retrogressive at best when ADS could have accomplished the same thing in a far more straightforward manner. After all, neither file type has any meaning apart from the associ-

ated directory or primary data stream anyway, so using ADS is the natural way to handle them. But, that would have meant that such information could not be shared with Windows platforms with FAT16 and FAT32 file systems. To hobble the OS is less costly than dealing with 40 million additional hits at the help desk.

But the most unfortunate aspect of ADS is that the negative press and exaggerated claims of vulnerability have polluted the waters to such a degree that the true potential of ADS may never be realized. ■

HAL BERGHEL (www.acm.org/hlb) is a professor and director of the School of Computer Science and director of the Center for Cybermedia Research at the University of Nevada, Las Vegas.

NATASA BRAJKOVSKA (natasa@crlmail.i2.nscce.edu) is a research assistant at the Center for Cybermedia Research at the University of Nevada, Las Vegas.

© 2004 ACM 0002-0782/04/0400 \$5.00