

A Primer on Reading Hex Dumps

A hexadecimal (“hex”) dump can be a very useful technique for examining a file, an application, or even the data on a drive. Some reasons for generating a hex dump can include:

- Debugging an application,
- Examining a corrupted file to determine if the file can be recovered,
- Figuring out what an unfamiliar protocol is doing,
- Reverse-engineering a program or malware,
- Verifying that the contents of a file really match the given file extension,
- Discovering data that has been placed in file slack after the end-of-file marker or in unallocated drive space. This technique can be very useful in forensics investigations.

Dumps often use hex (which is what we are going to discuss in this article), although octal dumps (which are used much less often) are also an option. Before starting the examination, determine the known facts about the file:

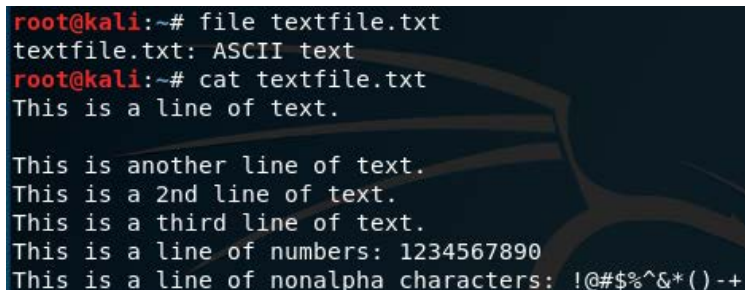
- Where did the file or dump come from?
- What is the purpose for dumping the file? For instance, if the file might be involved in a forensics investigation—STOP!—do not proceed to investigate the file on the *original* drive.
- Do you suspect that the file might be hiding steganography or using an incorrect file extension? Or could it contain one or more alternate data streams?

As with any project, it is necessary to understand the goal to choose the correct tool to use, and—to achieve the results you want—to really understand the tool you choose.

Hex dumps on a Linux system

Let’s start by using `file` to examine the file. If the file is text, we can `cat` it. If you `cat` a binary file, it may have control characters in the file that could make your system unstable.

On a Linux system, in many cases the file extension can be anything the user wants it to be. For Windows, the extension should match the file signature.

A terminal window with a dark background and light-colored text. The prompt is 'root@kali:~#'. The first command is 'file textfile.txt' and the output is 'textfile.txt: ASCII text'. The second command is 'cat textfile.txt' and the output is 'This is a line of text.' followed by a blank line, then 'This is another line of text.', 'This is a 2nd line of text.', 'This is a third line of text.', 'This is a line of numbers: 1234567890', and 'This is a line of nonalpha characters: !@#\$%^&*()-+'.

```
root@kali:~# file textfile.txt
textfile.txt: ASCII text
root@kali:~# cat textfile.txt
This is a line of text.

This is another line of text.
This is a 2nd line of text.
This is a third line of text.
This is a line of numbers: 1234567890
This is a line of nonalpha characters: !@#$%^&*()-+
```

Now, let’s look at the format of the *dump* and then we will consider the format of the *file*.

Format of the dump

We are going to show a dump file from each of the commands `hd`, `od`, and `xxd`, so you can see the differences between the three commands. Each command has pros and cons.

As shown in the figures below, a dump often has three sections.

- The **first vertical section** on the left is a column of incrementing numbers called the “offset,” usually shown in hex, but they can also be octal or even decimal numbers. The offset is the number of characters shown in each horizontal line of the dump. The hex offset is usually in

multiples of 4, 8, or 16, and the same offset multiple will be used throughout one dump. The samples show below are shown in hex in multiples of 16.

- The **second vertical section** shows the value of each character in hex. If you understand the patterns of the hex values of the characters, it is much easier to translate hex into ASCII and vice versa. Common pairs of hex values / ASCII characters include:
 - 20 (space),
 - 2e (period),
 - 0a (line feed - Linux),
 - 0D 0A (carriage return – Windows)
 - Numbers fall between 30 - 39,
 - Uppercase characters begin at 41 and run to 5A,
 - Lowercase characters start at 61 and reach 7A.
- The **third section** shows the ASCII representation of each hex value. Notice that ASCII values that cannot be represented (such as a line feed) are shown by dots. For the **od** command, the third section is the horizontal line below each line of the second section.

```
root@kali:~# hd textfile.txt
00000000  54 68 69 73 20 69 73 20  61 20 6c 69 6e 65 20 6f  |This is a line o|
00000010  66 20 74 65 78 74 2e 0a  0a 54 68 69 73 20 69 73  |f text...This is|
00000020  20 61 6e 6f 74 68 65 72  20 6c 69 6e 65 20 6f 66  | another line of|
00000030  20 74 65 78 74 2e 0a 54  68 69 73 20 69 73 20 61  | text..This is a|
00000040  20 32 6e 64 20 6c 69 6e  65 20 6f 66 20 74 65 78  | 2nd line of tex|
00000050  74 2e 0a 54 68 69 73 20  69 73 20 61 20 74 68 69  |t..This is a thi|
00000060  72 64 20 6c 69 6e 65 20  6f 66 20 74 65 78 74 2e  |rd line of text.|
00000070  0a 54 68 69 73 20 69 73  20 61 20 6c 69 6e 65 20  |.This is a line |
00000080  6f 66 20 6e 75 6d 62 65  72 73 3a 20 31 32 33 34  |of numbers: 1234|
00000090  35 36 37 38 39 30 0a 54  68 69 73 20 69 73 20 61  |567890.This is a|
000000a0  20 6c 69 6e 65 20 6f 66  20 6e 6f 6e 61 6c 70 68  | line of nonalph|
000000b0  61 20 63 68 61 72 61 63  74 65 72 73 3a 20 21 40  |a characters: !@|
000000c0  23 24 25 5e 26 2a 28 29  2d 2b 0a 0a                |#%^&*()-+...|
000000cc
```

No options are needed for the **hd** and **xxd** commands, but to make the output of **od** work the way we expect to read it (left to right), add **--endian=big**. Otherwise the digits will be reversed. For example, “5468” will display as “6854.” So the command reads:

```
od -Ax --endian=big -xc <filename>
```

The other options shown for the **od** command stand for:

-Ax: specifies hex output for file offsets.

-xc: 2-byte units of hex input are specified from column 2 and ASCII character output is displayed in the third column.

```
root@kali:~# od -Ax --endian=big -xc textfile.txt
000000  5468 6973 2069 7320 6120 6c69 6e65 206f  |T h i s   .   a l i n e |
000010  6620 7465 7874 2e0a 0a54 6869 7320 6973  |f t e x t . . a h i s |
000020  2061 6e6f 7468 6572 206c 696e 6520 6f66  | a n o t h e r   l i n e |
000030  2074 6578 742e 0a54 6869 7320 6973 2061  | t e x t . . T h i s   a |
000040  2032 6e64 206c 696e 6520 6f66 2074 6578  | 2 n d   l i n e   o f   t e x |
000050  742e 0a54 6869 7320 6973 2061 2074 6869  | t . . T h i s   a t h i |
000060  7264 206c 696e 6520 6f66 2074 6578 742e  | r d   l i n e   o f   t e x t . |
000070  0a54 6869 7320 6973 2061 206c 696e 6520  | . T h i s   a l i n e   |
000080  6f66 206e 756d 6265 7273 3a20 3132 3334  | o f   n u m b e r s :   1 2 3 4 |
000090  3536 3738 3930 0a54 6869 7320 6973 2061  | 5 6 7 8 9 0 . T h i s   a |
0000a0  206c 696e 6520 6f66 206e 6f6e 616c 7068  |   l i n e   o f   n o n a l p h |
0000b0  6120 6368 6172 6163 7465 7273 3a20 2140  | a   c h a r a c t e r s :   ! @ |
0000c0  2324 255e 262a 2829 2d2b 0a0a                | # $ % ^ & * ( ) - + \n \n |
0000cc
```

```

root@kali:~# xxd textfile.txt
00000000: 5468 6973 2069 7320 6120 6c69 6e65 206f  This is a line o
00000010: 6620 7465 7874 2e0a 0a54 6869 7320 6973  f text...This is
00000020: 2061 6e6f 7468 6572 206c 696e 6520 6f66  another line of
00000030: 2074 6578 742e 0a54 6869 7320 6973 2061  text..This is a
00000040: 2032 6e64 206c 696e 6520 6f66 2074 6578  2nd line of tex
00000050: 742e 0a54 6869 7320 6973 2061 2074 6869  t..This is a thi
00000060: 7264 206c 696e 6520 6f66 2074 6578 742e  rd line of text.
00000070: 0a54 6869 7320 6973 2061 206c 696e 6520  .This is a line
00000080: 6f66 206e 756d 6265 7273 3a20 3132 3334  of numbers: 1234
00000090: 3536 3738 3930 0a54 6869 7320 6973 2061  567890.This is a
000000a0: 206c 696e 6520 6f66 206e 6f6e 616c 7068  line of nonalph
000000b0: 6120 6368 6172 6163 7465 7273 3a20 2140  a characters: !@
000000c0: 2324 255e 262a 2829 2d2b 0a0a          #%^&*()-+..

```

Terms

A “file signature” is sometimes called a “magic number.” It is an identifier placed at a specific offset—usually 0—of a file.

A “file terminator” indicates the end of the file. It is sometimes called an EOF (end-of-file) marker. In section 2, there will be blanks after the EOF. An EOF marker can vary between file types, but it is sometimes represented by FF or 00 (sometimes called a “null byte”¹). Some EOF markers are listed in one of the magic number databases listed below.

A plain text file does not have a magic number, but it will have a EOF marker.

Notice the similarities—and the differences—between each hex dump program; especially how the end of the file is represented. Notice also how the dump program increments at the end of each file. **hd** and **od** show 0000 00cc, while **xxd** shows 0000 00c0. 0xc0 is the offset location where the file actually ends.

On a Windows system, a “data cluster” is the smallest amount of space that can be allocated to a file. The size of a data cluster, also known as an allocation unit, depends on the way a drive is subdivided and the operating system. The default size of an allocation unit in Windows is 4096 bytes.² Solid-state drives do not use clusters, so they operate a little differently. According to an article in ArsTechnica.com³, a typical modern SSD drive uses an 8196-byte page for reading and writing but may have to erase an entire block of data (256 pages) to re-write data.

Linux has several file systems to choose from, which include ext2, ext3 and ext4, XFS, JFS, ReiserFS and btrfs. My version of Kali uses Ext4⁴. Ext4 is a journaling file system that writes data in blocks. The size of the block will depend on the operating system and whether it is 32-bit or 64-bit, and the file system. The size of a block on my system is 4096 bytes. Blocks are then grouped into “block groups.”

Why would you want to know about clusters, page-sizes, and blocks? If you are looking for data hidden in file slack and unallocated drive space, you may *need* to know about them. In fact, if you *are* looking for hidden data, find out the exact specifications for the drive and the operating system that contains the data you are looking for. There should be no question—especially if you are conducting a forensic investigation—that you found hidden data in file slack or unallocated drive space.

¹ Wikipedia, “Null character,” https://en.wikipedia.org/wiki/Null_character

² How-To Geek, “What Should I Set the Allocation Unit Size to When Formatting?”

<https://www.howtogeek.com/136078/what-should-i-set-the-allocation-unit-size-to-when-formatting/>

³ Ars Technica, “Solid-state revolution: in-depth on how SSDs really work,” <https://arstechnica.com/information-technology/2012/06/inside-the-ssd-revolution-how-solid-state-disks-really-work/3/>

⁴ https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

Format of a file:

Depending on the type of file being examined, the layout of the file will differ. Take a JPG file for example. While there are different versions of JPG that can be used, we will examine the format of a standard called "JPEG/JFIF." A JPEG image format (shown below) is a complicated structure that starts with FFD8 and ends with FFD9.

JFIF file structure		
Segment	Code	Description
SOI	FF D8	Start of Image
JFIF-APP0	FF E0 s1 s2 4A 46 49 46 00 ...	see below
JFXX-APP0	FF E0 s1 s2 4A 46 58 58 00 ...	optional, see below
... additional marker segments (for example SOF, DHT, COM)		
SOS	FF DA	Start of Scan
	compressed image data	
EOI	FF D9	End of Image

Figure shown from https://en.wikipedia.org/wiki/JPEG_File_Interchange_Format

First, let's again use **file** to find out some information about sunset.jpg.

```
root@kali:~/Pictures# file sunset.jpg
sunset.jpg: JPEG image data, JFIF standard 1.02, resolution (DPI), density 96x96
, segment length 16, Exif Standard: [TIFF image data, big-endian, direntries=13,
description=Lightning storm, orientation=upper-left, xresolution=186, yresoluti
on=194, resolutionunit=2, software=Adobe Photoshop CS3 Windows, datetime=2015:05
:18 08:36:23], baseline, precision 8, 1920x1200, frames 3
```

Next, if you look at screenshots of the start and end of a JPG file dump, you can see the same elements in the dump are shown in the file layout above. You can also see "JFIF" in the third section of the dump. That also matches the layout of a JPG file as shown above.

```
root@kali:~/Pictures# hd sunset.jpg
00000000 ff d8 ff e0 00 10 4a 46 49 46 00 01 02 01 00 60 |.....JFIF.....|
00000010 00 00 00 00 ff e1 21 0c 45 78 69 66 00 00 4d 4d |.!.Exif..MM|
00000020 00 2a 00 00 00 08 00 0d 01 0e 00 02 00 00 10 |.*.....|
00000030 00 00 00 aa 01 12 00 03 00 00 00 01 00 01 00 00 |.....|
00000040 01 1a 00 05 00 00 00 01 00 00 00 ba 01 1b 00 05 |.....|
00000050 00 00 00 01 00 00 00 c2 01 28 00 03 00 00 00 01 |.....(|
00000060 00 02 00 00 01 31 00 02 00 00 00 1c 00 00 00 ca |....1.....|
00000070 01 32 00 02 00 00 00 14 00 00 00 e6 01 3b 00 02 |.2.....;..|
00000080 00 00 00 1c 00 00 00 fa 82 98 00 02 00 00 00 1c |.....|
00000090 00 00 01 16 9c 9b 00 01 00 00 00 20 00 00 01 32 |.....2|
000000a0 9c 9d 00 01 00 00 00 38 00 00 01 52 ea 1c 00 07 |.....8...R...|
000000b0 00 00 07 c4 00 00 01 8a 87 69 00 04 00 00 00 01 |.....i.....|
```

Beginning of JPG file dump; notice "FF D8," which is the start of a JPG, while "FF E0" is the start of JFIF-APP0.

```
000c54c0 6a 73 0e 2e 71 1f e5 58 6d 44 94 f2 cf 47 4a 7c |js..q..XmD...GJ|
000c54d0 b4 62 66 7a 68 3c c9 a6 32 d4 38 89 15 a6 2a 2c |.bfzh<...2.8...*,|
000c54e0 4b db b7 0d 99 92 48 10 24 cc 0e af 67 1c 68 fd |K....H.$...g.h.|
000c54f0 cb e2 ea 62 12 90 4c 9d 22 24 89 82 7d a6 00 81 |...b..L."$.}...|
000c5500 8e ca 36 b5 91 52 55 d5 0a ac 2c 05 93 41 21 51 |..6..RU....A!Q|
000c5510 a3 03 f0 3c 37 46 1b 68 99 d8 5e ca 62 c4 b2 56 |...<7F.h...^..b..V|
000c5520 5c c7 b1 59 71 1a d5 b4 ec 03 33 da d7 6b 76 25 |\.Yq.....3..kv%|
000c5530 6d f9 f3 40 68 d9 49 16 9d 47 6c 57 ff d9 |m..@h.I..GLW..|
000c553e
```

Tail end of JPG file dump; notice "FF D9."

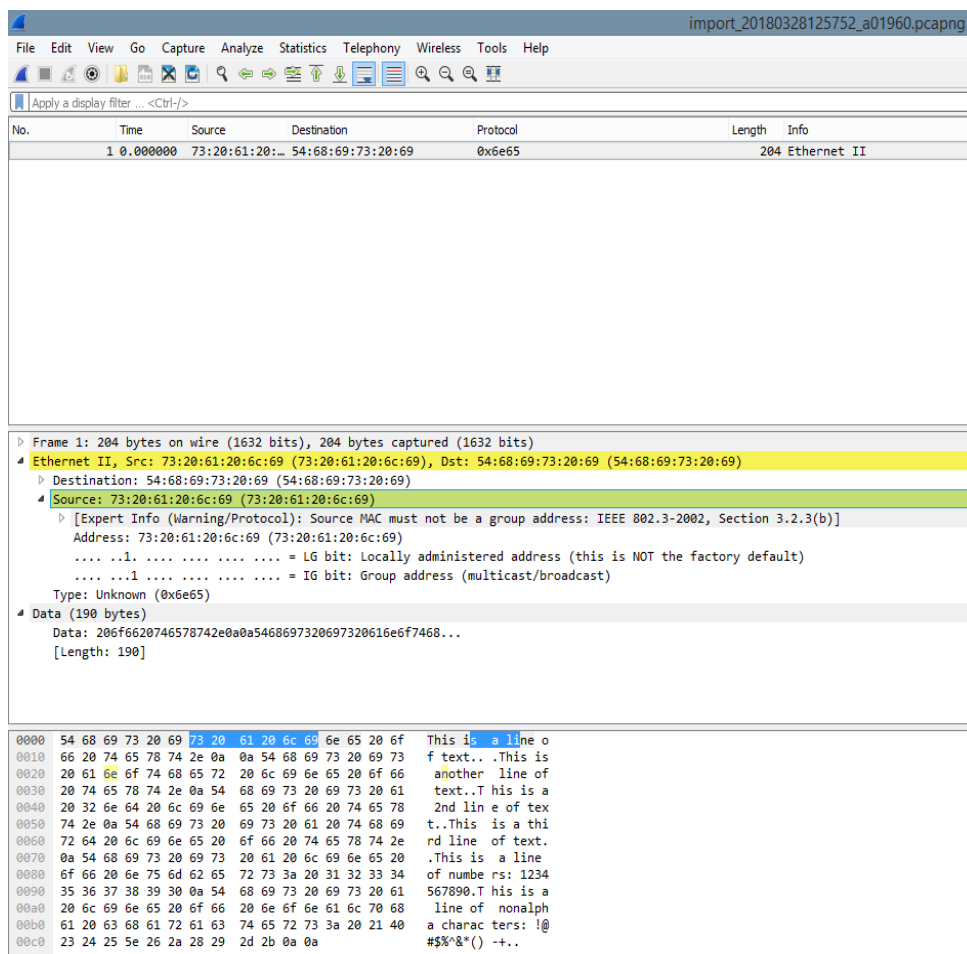
The file ends with “FF D9” just as it is supposed to with no additional data after the EOF marker. This file is a JPG, just as we thought.

Wireshark

Wireshark is a tool that can be used on either Windows or Linux systems. A dump can be imported into Wireshark⁵ if it has been generated by using the following command: `od -Ax -tx1 -v`. The dump will look like this:

```
root@kali:~# od -Ax -tx1 -v textfile.txt > wireshark.txt
root@kali:~# cat wireshark.txt
000000 54 68 69 73 20 69 73 20 61 20 6c 69 6e 65 20 6f
000010 66 20 74 65 78 74 2e 0a 0a 54 68 69 73 20 69 73
000020 20 61 6e 6f 74 68 65 72 20 6c 69 6e 65 20 6f 66
000030 20 74 65 78 74 2e 0a 54 68 69 73 20 69 73 20 61
000040 20 32 6e 64 20 6c 69 6e 65 20 6f 66 20 74 65 78
000050 74 2e 0a 54 68 69 73 20 69 73 20 61 20 74 68 69
000060 72 64 20 6c 69 6e 65 20 6f 66 20 74 65 78 74 2e
000070 0a 54 68 69 73 20 69 73 20 61 20 6c 69 6e 65 20
000080 6f 66 20 6e 75 6d 62 65 72 73 3a 20 31 32 33 34
000090 35 36 37 38 39 30 0a 54 68 69 73 20 69 73 20 61
0000a0 20 6c 69 6e 65 20 6f 66 20 6e 6f 6e 61 6c 70 68
0000b0 61 20 63 68 61 72 61 63 74 65 72 73 3a 20 21 40
0000c0 23 24 25 5e 26 2a 28 29 2d 2b 0a 0a
0000cc
root@kali:~#
```

The “Import from Hexdump” command will be found under File in the Wireshark menu bar. A typical dump that has been imported into Wireshark will resemble the following screenshot. It is a convenient way to examine a large file.



⁵ Wireshark.org, “5.5. Import hex dump,”

https://www.wireshark.org/docs/wsug_html_chunked/ChIOImportSection.html

Hex dumps on a Windows system

PowerShell

If you have Windows 10 and PowerShell 5 (which is installed by default in 10) or have downloaded and installed version 6⁶ (PowerShell 6 can also be installed on Linux⁷), generating a hex dump is very easy. Type **Format-Hex .\textfile.txt** and you will see the following:

```
PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Format-Hex .\sunset.jpg

Path: C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\sunset.jpg

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000 FF D8 FF E0 00 10 4A 46 49 46 00 01 02 01 00 60 .ø.à..JFIF.....
00000010 00 60 00 00 FF E1 21 0C 45 78 69 66 00 00 4D 4D .`...á!.Exif..MM
00000020 00 2A 00 00 00 08 00 0D 01 0E 00 02 00 00 10 .*.
00000030 00 00 00 AA 01 12 00 03 00 00 00 01 00 01 00 00 ...ª.....
00000040 01 1A 00 05 00 00 00 01 00 00 00 BA 01 1B 00 05 .....º.....
00000050 00 00 00 01 00 00 00 C2 01 28 00 03 00 00 01 .....Ä.(.....
00000060 00 02 00 00 01 31 00 02 00 00 00 1C 00 00 00 CA .....1.....Ë
00000070 01 32 00 02 00 00 00 14 00 00 00 E6 01 3B 00 02 .2.....æ;..
00000080 00 00 00 1C 00 00 00 FA 82 98 00 02 00 00 00 1C .....ü.....
00000090 00 00 01 16 9C 9B 00 01 00 00 00 20 00 00 01 32 .....
000000A0 00 9C 9D 00 01 00 00 00 38 00 00 01 52 EA 1C 00 07 .....8...Rê...
000000B0 00 00 07 C4 00 00 01 8A 87 69 00 04 00 00 00 01 ...Ä...i.....
```

If you look at a hex dump of the same file on both a Windows and a Linux system, each dump should display the same character values in the third column if both programs are using the same character encoding scheme (i.e., ASCII or Unicode). There is one exception: Windows uses a carriage return line feed (0D 0A) while Linux only uses a linefeed (0A) at the end of a line. On Linux, a dump of a text file using any of the commands previously shown will have much the same content, although the format of the dump will differ slightly between commands used. But, when you look at the PowerShell dump screenshots of sunset.jpg (below), you will notice that the characters shown in section 3 of the dump are *not* the same as the Linux dumps.

ASCII⁸ is used as a base for most character encoding schemes, but it only has space for 128 characters. In decimal, that is depicted as 0 to 127, while hex values run from 0 to 7F. According to Wikipedia⁹, Unicode version 10.0 contains over 136,000 characters and covers multiple languages (examples include Latin, Greek, and Mongolian) and symbol sets (such as mathematical, currency, dingbats, and arrows).

```
PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Format-Hex .\sunset.jpg

Path: C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\sunset.jpg

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000 FF D8 FF E0 00 10 4A 46 49 46 00 01 02 01 00 60 .ø.à..JFIF.....
00000010 00 60 00 00 FF E1 21 0C 45 78 69 66 00 00 4D 4D .`...á!.Exif..MM
00000020 00 2A 00 00 00 08 00 0D 01 0E 00 02 00 00 10 .*.
00000030 00 00 00 AA 01 12 00 03 00 00 00 01 00 01 00 00 ...ª.....
00000040 01 1A 00 05 00 00 00 01 00 00 00 BA 01 1B 00 05 .....º.....
00000050 00 00 00 01 00 00 00 C2 01 28 00 03 00 00 01 .....Ä.(.....
00000060 00 02 00 00 01 31 00 02 00 00 00 1C 00 00 00 CA .....1.....Ë
00000070 01 32 00 02 00 00 00 14 00 00 00 E6 01 3B 00 02 .2.....æ;..
00000080 00 00 00 1C 00 00 00 FA 82 98 00 02 00 00 00 1C .....ü.....
00000090 00 00 01 16 9C 9B 00 01 00 00 00 20 00 00 01 32 .....
000000A0 00 9C 9D 00 01 00 00 00 38 00 00 01 52 EA 1C 00 07 .....8...Rê...
000000B0 00 00 07 C4 00 00 01 8A 87 69 00 04 00 00 00 01 ...Ä...i.....
```

⁶ Microsoft, "PowerShell,"

<https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view=powershell-6>

⁷ Campbell, Steven, "Install PowerShell on Kali Linux,"

<https://www.stevencampbell.info/2017/04/install-powershell-on-kali-linux/>

⁸ AsciiTable, "ASCII table and description," <http://www.asciitable.com/>

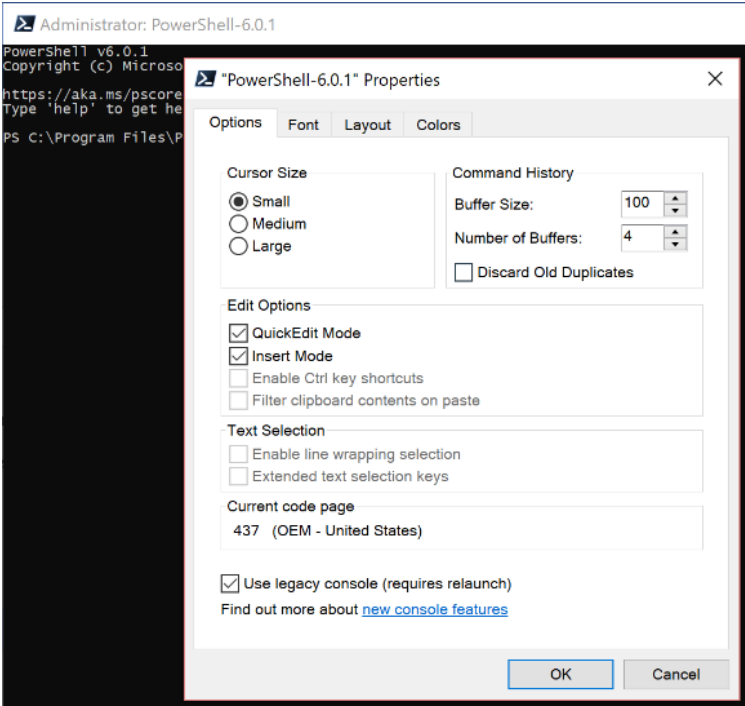
⁹ Wikipedia, "List of Unicode characters," https://en.wikipedia.org/wiki/List_of_Unicode_characters

```
000C5440  F8 F8 B9 B8 58 A4 45 25 14 1C 3E 09 59 81 D5 18 0q1,XuE%...>.YmÖ.
000C5450  6B 90 7C A3 AD D7 CD 8C 95 F8 ED 6B 69 F0 E5 4A k|E-xIÖöíkiðäJ
000C5460  01 AF 26 41 A7 A9 30 2C 1B 15 A0 F2 E0 2A 88 08 .&A$00,... öä*Ö.
000C5470  65 75 D4 6B A8 D7 E9 F0 3C 46 1B 49 A5 E7 1A C8 euÖk"xëö<F.IYç.È
000C5480  F8 1D 26 21 83 54 E0 19 9A 21 53 48 5F 14 90 CA ø.&!ÖTä.Ö!SK_.ÖÈ
000C5490  50 B2 92 92 29 46 04 C6 41 17 52 75 04 30 F0 3C PZÖÖ)F.ÆA.Ru.Öö<
000C54A0  7B 4F 01 45 09 25 26 4D 03 7D 11 F4 85 D2 5E 87 {O.E.%&M.}.ööÖ^Ö
000C54B0  67 9A 1E A4 CD 99 33 0E 6E C6 30 BC 26 A3 02 C1 gÖ.uIÖ3.næÖ&E.Ä
000C54C0  6A 73 0E 2E 71 1F E5 58 6D 44 94 F2 CF 47 4A 7C js..q.&XmDÖbIGJ|
000C54D0  B4 62 66 7A 68 3C C9 A6 32 D4 38 89 15 A6 2A 2C 'bfzh<È;ZÖBÖ.}|*,
000C54E0  4B DB 87 0D 99 92 48 10 24 CC 0E AF 67 1C 68 FD KÜ..ÖÖH.$i.~g.hy
000C54F0  CB E2 EA 62 12 90 4C 9D 22 24 89 82 7D A6 00 81 Èäëb.ÖLÖ"$ÖÖÖ}|.Ö
000C5500  8E CA 36 B5 91 52 55 D5 0A AC 2C 05 93 41 21 51 ÖÈgµÖRUÖ.~,.ÖA!Q
000C5510  A3 03 F0 3C 37 46 1B 68 99 D8 5E CA 62 C4 B2 56 E.ö<7F.hZÖ^ÈbA^V
000C5520  5C C7 B1 59 71 1A D5 B4 EC 03 33 DA D7 6B 76 25 \Ç+Yq.Ö'1.3Ü×kv%
000C5530  6D F9 F3 40 68 D9 49 16 9D 47 6C 57 FF D9 müö@hÜI.ÖGLW.U

PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> ^[[?1;0c^[?1;0c^[?1;0c^[?1;0c^[?1;0c^[?1;0c^[?1;0c^[?1;0c
```

When we look at the beginning of the PowerShell hex dump, we see it has a problem displaying some characters at offset A0. The tail end of the JPG hex dump definitely does not end as gracefully as the Linux built-in commands do. That is because PowerShell versions 5 and 6 use Unicode for character display.

You can fix the display issue to a certain extent by changing the screen settings: Open PowerShell 5/6 in admin mode, click on the PowerShell icon in the upper left-hand corner, choose “Properties” and select “Use legacy console¹⁰” I also had to change Font and Layout sizes because the default screen and font sizes were so small, they were almost unusable. You can change the settings back to the original by again clicking on the PowerShell icon and choosing “Defaults” from the dropdown menu. A change to legacy console settings will require closing and re-opening PowerShell.



This does fix the display issue, as shown below. Just note that while the hex should be almost the same no matter which program or operating system you use, what is shown in the third section may be different, because the program could be using either ASCII or Unicode to display characters. Since Unicode is based on ASCII (the first 128 characters of both are the same), if the symbols for the hex

¹⁰ Windows 10 Forums, “Windows 10: Enable or disable legacy console for Command Prompt and PowerShell in Windows 10,” <https://www.tenforums.com/tutorials/94146-enable-disable-legacy-console-cmd-powershell-windows-10-a.html>

values used are within the ASCII set, the characters in the third column will be much the same, except for line returns. Therefore, a hex dump of a text file should be similar on both systems.

```
PowerShell-6.0.1
PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> $PSVersionTable.PSVersion
Major Minor Patch PreReleaseLabel BuildLabel
-----
6 0 1

PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Format-Hex .\textfile.txt

Path: C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile.txt
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 54 68 69 73 20 69 73 20 61 20 6C 69 6E 65 20 6F This is a line o
00000010 66 20 74 65 78 74 2E 0D 0A 0D 0A 54 68 69 73 20 f text....This
00000020 69 73 20 61 6E 6F 74 68 65 72 20 6C 69 6E 65 20 is another line
00000030 6F 66 20 74 65 78 74 2E 0D 0A 54 68 69 73 20 69 of text...This i
00000040 73 20 61 20 32 6E 64 20 6C 69 6E 65 20 6F 66 20 s a 2nd line of
00000050 74 65 78 74 2E 0D 0A 54 68 69 73 20 69 73 20 61 text...This is a
00000060 20 74 68 69 72 64 20 6C 69 6E 65 20 6F 66 20 74 third line of t
00000070 65 78 74 2E 0D 0A 54 68 69 73 20 69 73 20 61 20 ext...This is a
00000080 6C 69 6E 65 20 6F 66 20 6E 75 6D 62 65 72 73 3A line of numbers:
00000090 20 31 32 33 34 35 36 37 38 39 30 0D 0A 54 68 69 1234567890..Thi
000000A0 73 20 69 73 20 61 20 6C 69 6E 65 20 6F 66 20 6E s is a line of n
000000B0 6F 6E 61 6C 70 68 61 20 63 68 61 72 61 63 74 65 onalpa characte
000000C0 72 73 3A 20 21 40 23 24 25 5E 26 2A 28 29 2D 2B rs: !@#%&*()-+

PS C:\users\Kalmia11\Documents\Classes\AttackingNetworkProtocols>
```

But when you examine a JPG file using PowerShell to generate the hex dump, you can clearly see it is using Unicode for character encoding in the third column.

```
PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> $PSVersionTable.PSVersion
Major Minor Build Revision
-----
5 1 16299 251

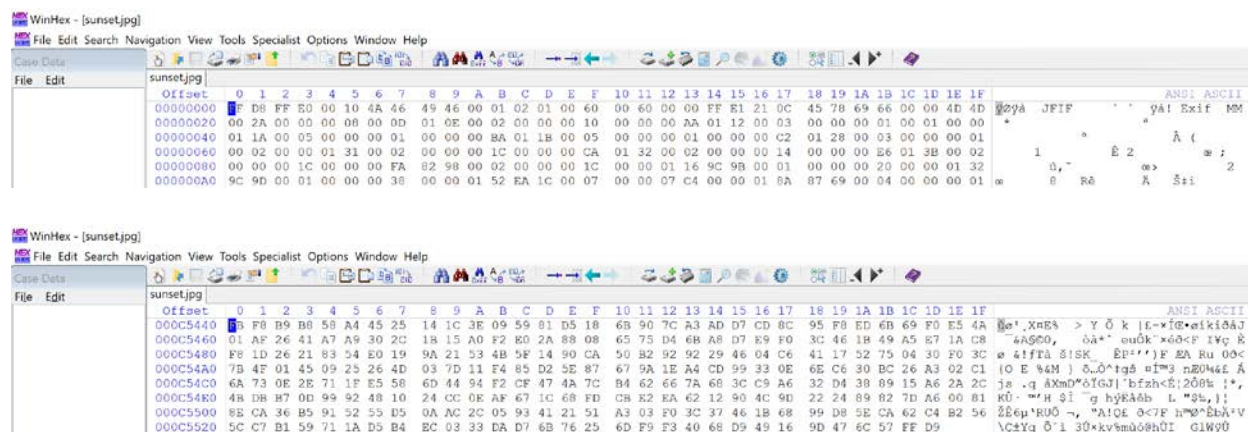
PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> format-hex .\sunset.jpg

Path: C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\sunset.jpg
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 FF D8 FF E0 00 10 4A 46 49 46 00 01 02 01 00 60 .ø.à..JFIF....`
00000010 00 60 00 00 FF E1 21 0C 45 78 69 66 00 00 4D 4D .`....á!.Exif..MM
00000020 00 2A 00 00 00 08 00 0D 01 0E 00 02 00 00 00 10 .*.
00000030 00 00 00 AA 01 12 00 03 00 00 00 01 00 01 00 00 ...ª.....
00000040 01 1A 00 05 00 00 00 01 00 00 00 BA 01 1B 00 05 .....º.....
00000050 00 00 00 01 00 00 00 C2 01 28 00 03 00 00 00 01 .....Ä.(.....
00000060 00 02 00 00 01 31 00 02 00 00 00 1C 00 00 00 CA .....1.....Ê
00000070 01 32 00 02 00 00 00 14 00 00 00 E6 01 3B 00 02 .2.....æ;..
00000080 00 00 00 1C 00 00 00 FA 82 98 00 02 00 00 00 1C .....ú.....
00000090 00 00 01 16 9C 9B 00 01 00 00 00 20 00 00 01 32 ..... ..2
000000A0 9C 9D 00 01 00 00 00 38 00 00 01 52 EA 1C 00 07 .....8...Ré...
000000B0 00 00 07 C4 00 00 01 8A 87 69 00 04 00 00 00 01 ...Ä...i.....

000C5480 F8 1D 26 21 83 54 E0 19 9A 21 53 4B 5F 14 90 CA ø.&!Tà.¡SK_.Ê
000C5490 50 B2 92 92 29 46 04 C6 41 17 52 75 04 30 F0 3C P²)F.ÆA.Ru.0ð<
000C54A0 7B 4F 01 45 09 25 26 4D 03 7D 11 F4 85 D2 5E 87 {O.E.%&M.}.øÒ^
000C54B0 67 9A 1E A4 CD 99 33 0E 6E C6 30 BC 26 A3 02 C1 g.ªÍ3.nÆ0%&Æ.Á
000C54C0 6A 73 0E 2E 71 1F E5 58 6D 44 94 F2 CF 47 4A 7C js..q.âXmDðöİGJ|
000C54D0 B4 62 66 7A 68 3C C9 A6 32 D4 38 89 15 A6 2A 2C `bfzh<É!208.!.*,
000C54E0 4B DB B7 0D 99 92 48 10 24 CC 0E AF 67 1C 68 FD KÜ..H.$Î.°g.hý
000C54F0 CB E2 EA 62 12 90 4C 9D 22 24 89 82 7D A6 00 81 Êâéb.QL"$}!.
000C5500 8E CA 36 B5 91 52 55 D5 0A AC 2C 05 93 41 21 51 ÊÊ6µRUÖ.¬,.A!Q
000C5510 A3 03 F0 3C 37 46 1B 68 99 D8 5E CA 62 C4 B2 56 £.ð<7F.hðø^ÊbÄ²V
000C5520 5C C7 B1 59 71 1A D5 B4 EC 03 33 DA D7 6B 76 25 \Ç±Yq.ð`i.3Úxkv%
000C5530 6D F9 F3 40 68 D9 49 16 9D 47 6C 57 FF D9 mùó@hÛI.ÆGlW.Û
```


WinHex

WinHex¹¹ is another hex dump application available to Windows users. It is definitely easy to use. It is available in a free evaluation and has several types of paid licenses. Features include the ability to scroll up and down through the file and make changes to the data. If you wish to *edit* a large file and save the changes, buy WinHex.



Alternate Data Streams

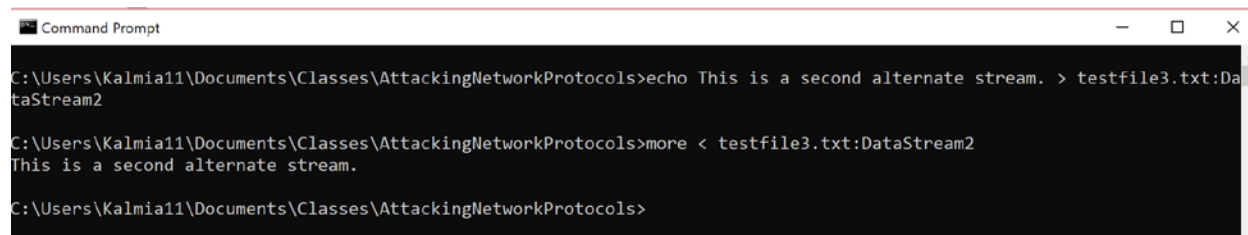
Several years ago, Robert Mitchell wrote a blog on Microsoft TechNet called “Alternate Data Streams in NTFS.” His definition of an alternate data stream (ADS) is that “it is a data stream that is alternate.” ADS is not available on a FAT system.

On Windows, a file has several attributes, one of which is called \$DATA. It is also known as the “data attribute.” The data attribute is where the data, or the text, of a file is stored. This data attribute is also called the “unnamed data stream.” If a data stream is *named*, then it is an alternate data stream. There can be many alternate data streams attached to a file or even a directory.

Why do we care about ADS? Well, it is a great way to hide information in a file or in a directory, such as malware. It is also used by the Windows system to track information about a file. Some well-known named streams used by Windows are SummaryInformation, DocumentSummaryInformation, and Zone.Identifier.

There are several ways to create a file with ADS. One way is at a command prompt.

- To create an ADS file, type:
`echo This is a second alternate stream. > testfile3.txt:DataStream2`
- To see the contents of an ADS file, type:
`more < testfile3.txt:DataStream2`



One way to delete the ADS information from a file is to copy the file to a FAT drive and copy it back to the NTFS system

¹¹ X-Ways, “WinHex: Computer Forensics & Data Recovery Software, Hex Editor & Disk Editor,” <https://www.x-ways.net/winhex/>

You can also create a file with alternate data streams using PowerShell.

- To check for an alternate data stream in a file, type:
`Get-Item -Path .\textfile2.txt -Stream *`
- To create an alternate data stream in a file, type:
`Set-Content -Path .\textfile2.txt -Stream DataStream2`
- To see the contents of an alternate data stream, type:
`Get-Content -Path .\textfile2.txt -Stream DataStream2`
- To remove an alternate data stream, type:
`Remove-Item -Path .\textfile2.txt -Stream DataStream2`

```
Windows PowerShell
PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Set-Content -Path .\textfile2.txt -Stream DataStream2

cmdlet Set-Content at command pipeline position 1
Supply values for the following parameters:
Value[0]: This is the second alternate stream.
Value[1]:
PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> get-item -Path .\textfile2.txt -Stream *

PSPath           : Microsoft.PowerShell.Core\FileSystem::C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile2.txt::$DATA
PSParentPath     : Microsoft.PowerShell.Core\FileSystem::C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols
PSChildName      : textfile2.txt::$DATA
PSDrive          : C
PSProvider       : Microsoft.PowerShell.Core\FileSystem
PSIsContainer    : False
FileName        : C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile2.txt
Stream           :::$DATA
Length          : 208

PSPath           : Microsoft.PowerShell.Core\FileSystem::C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile2.txt:DataStream2
PSParentPath     : Microsoft.PowerShell.Core\FileSystem::C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols
PSChildName      : textfile2.txt:DataStream2
PSDrive          : C
PSProvider       : Microsoft.PowerShell.Core\FileSystem
PSIsContainer    : False
FileName        : C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile2.txt
Stream           : DataStream2
Length          : 38

PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Get-Content -Path .\textfile2.txt -Stream DataStream2
This is the second alternate stream.
```

You can use a Microsoft Sysinternals tool¹² called Streams.exe to discover if a file or a directory has extra data streams, but Streams will only display the names and sizes of named streams found. One advantage of using Streams is that it can search an entire drive for files containing ADS, although it will take several hours.

```
C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols>streams testfile3.txt

streams v1.60 - Reveal NTFS alternate streams.
Copyright (C) 2005-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\testfile3.txt:
:DataStream2:$DATA 37
```

¹² Sysinternals Suite, <https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite>

The funny thing is that if you generate a hex dump for a file that has an ADS, you will not see any indication of the existence of alternate data streams (see below).

```
PS C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols> Format-Hex .\textfile2.txt

Path: C:\Users\Kalmia11\Documents\Classes\AttackingNetworkProtocols\textfile2.txt

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  54 68 69 73 20 69 73 20 61 20 6C 69 6E 65 20 6F  This is a line o
00000010  66 20 74 65 78 74 2E 0D 0A 0D 0A 54 68 69 73 20  f text....This
00000020  69 73 20 61 6E 6F 74 68 65 72 20 6C 69 6E 65 20  is another line
00000030  6F 66 20 74 65 78 74 2E 0D 0A 54 68 69 73 20 69  of text...This i
00000040  73 20 61 20 32 6E 64 20 6C 69 6E 65 20 6F 66 20  s a 2nd line of
00000050  74 65 78 74 2E 0D 0A 54 68 69 73 20 69 73 20 61  text...This is a
00000060  20 74 68 69 72 64 20 6C 69 6E 65 20 6F 66 20 74  third line of t
00000070  65 78 74 2E 0D 0A 54 68 69 73 20 69 73 20 61 20  ext...This is a
00000080  6C 69 6E 65 20 6F 66 20 6E 75 6D 62 65 72 73 3A  line of numbers:
00000090  20 31 32 33 34 35 36 37 38 39 30 0D 0A 54 68 69  1234567890..Thi
000000A0  73 20 69 73 20 61 20 6C 69 6E 65 20 6F 66 20 6E  s is a line of n
000000B0  6F 6E 61 6C 70 68 61 20 63 68 61 72 61 63 74 65  onalpha characte
000000C0  72 73 3A 20 21 40 23 24 25 5E 26 2A 28 29 2D 2B  rs: !@#$%^&*()-+
```

Steganography

What is steganography? According to Black Slash in [“How to Hide Secret Data Inside an Image or Audio File in Seconds,”](#) “steganography is the art of hiding information in plain sight.” Various steganography techniques have been used since at least the time of Herodotus, a Greek historian during the fifth century BC, who told the story of a message inscribed on the shaved head of a slave. After the slave’s hair grew long enough to obscure the message, he was sent to Greece to warn of Persian invaders.

For this demonstration, we will use a program called Steghide,¹³ which has been around since 2003. It uses “a graph-theoretic (sic) approach to steganography.” It is amazingly easy to hide secret data of any format in JPG, BMP, WAV and AU files—called a “cover file”—with a program like Steghide.

However, Steghide cannot hide information in a text file, although it can hide a text file in the cover file. There are a couple reasons for this:

1. Text files are usually smaller size files.
2. Small changes to a text file will probably be very obvious because changing one bit completely changes the character. For example, change 0x71 to 0x70 and the word, “square” becomes “spuare.” Changing one bit of a color does not affect the color of a JPEG dramatically.

How easy is it to detect whether a picture has been altered to include a message? That depends.

1. Do you have an idea of the program (such as Steghide and OpenStego) used to hide the data? Each steganography program has pluses and minuses that will have to be considered before a choice is made to hide or try to recover the secret message.
2. Do you have the original file to compare to the file that you think contains a hidden message? One indication that the file could contain a hidden file is that the size of the file may be either slightly larger or smaller than the original.

Here is how to use Steghide to add the file “secret.txt” to sunset.jpg:

```
steghide --embed -ef secret.txt -cf sunset.jpg -sf sunset-HD-2.jpg -e none -Z
```

¹³ Steghide, <http://steghide.sourceforge.net/index.php>

```

root@kali:~/hexdump# steghide --embed -ef secret.txt -cf sunset.jpg -sf sunset-HD-2.jpg -e none -Z
Enter passphrase:
Re-Enter passphrase:
embedding "secret.txt" in "sunset.jpg"... done
writing stego file "sunset-HD-2.jpg"... done
root@kali:~/hexdump# ls -l sunset*.
-rw-r--r-- 1 root root 773901 Apr 25 22:00 sunset-HD-2.jpg
-rw----- 1 root root 808254 Jun 20 2015 sunset.jpg
root@kali:~/hexdump#

```

Only the simplest version of the command has been used here, but there are also options for compressing and encrypting the hidden data.

Note the difference in size between the JPG with the embedded file and the JPG without the file. The size of the JPG with the embedded file actually decreases! Notice also that the file information changes.

```

root@kali:~/hexdump# file sunset.jpg
sunset.jpg: JPEG image data, JFIF standard 1.02, resolution (DPI), density 96x96, segment length 16, Exif Stand
rd: [TIFF image data, big-endian, direntries=13, description=Lightning storm, orientation=upper-left, xresolutio
n=186, yresolution=194, resolutionunit=2, software=Adobe Photoshop CS3 Windows, datetime=2015:05:18 08:36:23], b
aseline, precision 8, 1920x1200, frames 3
root@kali:~/hexdump# file sunset-HD-2.jpg
sunset-HD-2.jpg: JPEG image data, JFIF standard 1.02, resolution (DPI), density 96x96, segment length 16, baseli
ne, precision 8, 1920x1200, frames 3
root@kali:~/hexdump#

```

The hex dump of the new JPG also has subtle changes from the original. Notice how this file ends at xBCF0D, where the original file ended at xC5530.

```

000bce80  c4 9e 21 f1 76 b1 a5 e9 37 1a 16 8b 73 e2 1d 5c |...!.v...7...s..\|
000bce90  ea 3f d9 5a 6d c4 96 f2 cf 67 6a 7c b4 62 66 7b |.?.Zm....gj|.bf{|
000bcea0  68 3c c9 a6 32 dc 38 89 15 a6 2a 30 7c fa 78 75 |h<..2.8...*0|.xu|
000bceb0  4d de ed b4 ac ae ef 65 e5 f7 75 d4 f7 ea 63 9d |M.....e..u...c.|
000bcec0  58 da d1 8a 6e ef 95 5a ed 5e cd fd ee c9 59 6b |X...n..Z.^....Yk|
000bced0  b1 f5 ad e4 56 97 77 42 eb 4b 01 64 e0 48 54 70 |...V.wB.K.d.HTp|
000bcee0  c0 fb 1a f5 e1 a6 e7 8d 56 d3 d8 c2 d4 bc 15 e1 |.....V.....|
000bcef0  cd 7b 55 97 51 bd 5c 4e c0 33 3e 31 96 c7 42 57 |.{U.Q.\N.3>1..BW|
000bcf00  1f ad 24 b9 36 39 27 1e 67 bd 8f ff d9 |..$.69'.g....|
000bcf0d

```

While you can extract the secret file just as easily, there does not appear to be a way to *remove* the secret file from the cover file.

```
steghide extract -sf sunset-HD-2.jpg -xf sunset-clean.jpg
```

Hiding information in the slack space at the end of a file

This technique does not appear to be used much anymore. I could not find any tools except **bmap**¹⁴ (which is not supported anymore) that write in file slack space. It is only available for Linux.

I had to install the **gcc-multilib** package before I could compile **bmap** in Kali. It still had warnings, but I chose to ignore them since **bmap** worked. **bmap** is very easy to use.

This first command shown below uses the **checkslack** option to test the file for content in the slack space. The **map** option lists the sector numbers where the file is saved:

```

bmap --mode checkslack textfile.txt --verbose
bmap --mode map textfile.txt --verbose

```

¹⁴ CameronLonsdale/bmap, <https://github.com/CameronLonsdale/bmap>


```

root@kali:~/hexdump# bmap --mode checkslack textfile.txt --verbose
textfile.txt does not have slack
root@kali:~/hexdump# bmap --mode map textfile.txt --verbose
54822128
54822129
54822130
54822131
54822132
54822133
54822134
54822135

```

The **putslack** option adds the data to be hidden in the file slack space. As you can see in the listing below, there is no change in the size of the files between textfile.txt and textfile-2.txt.

```

root@kali:~/hexdump# echo "secret" | bmap --mode putslack textfile-2.txt --verbose stuffing
stuffing block 40915
file size was: 204
slack size: 3892
block size: 4096
root@kali:~/hexdump# ls -l
total 4284
-rw-r--r-- 1 root root    173 Apr 23 16:35 secret.txt
-rw-r--r-- 1 root root    173 Apr 25 23:19 sunset-clean.jpg
-rw-r--r-- 1 root root 773901 Apr 25 22:00 sunset-HD-2.jpg
-rw----- 1 root root 808254 Jun 20  2015 sunset.jpg
-rw-r--r-- 1 root root    204 Apr 25 23:22 textfile-2.txt
-rw-r--r-- 1 root root    204 Mar 28 12:30 textfile.txt
-rw-r--r-- 1 root root 2778381 Mar 28 13:18 wireshark-pic
-rw-r--r-- 1 root root    710 Apr 22 18:54 wireshark.txt
root@kali:~/hexdump# bmap --mode slack textfile-2.txt --verbose
getting from block 40915
file size was: 204
slack size: 3892
block size: 4096
secret

```

Next, we try various tests on the file containing the hidden data to see if the data can be easily found. The hidden data does not show up. Obviously if we did a true forensic examination of the file, the data would be discovered, as forensic tools can examine every byte of the drive space.

```

root@kali:~/hexdump# grep secret textfile-2.txt
root@kali:~/hexdump# grep 1234 textfile-2.txt
This is a line of numbers: 1234567890
root@kali:~/hexdump# strings textfile-2.txt
This is a line of text.
This is another line of text.
This is a 2nd line of text.
This is a third line of text.
This is a line of numbers: 1234567890
This is a line of nonalpha characters: !@#$%^&*()-+

```

What I found most surprising is that data written in the file slack space does not show up in a hex dump of the file.

```

root@kali:~/hexdump# hd textfile-2.txt
00000000  54 68 69 73 20 69 73 20  61 20 6c 69 6e 65 20 6f  |This is a line o|
00000010  66 20 74 65 78 74 2e 0a  0a 54 68 69 73 20 69 73  |f text...This is|
00000020  20 61 6e 6f 74 68 65 72  20 6c 69 6e 65 20 6f 66  |another line of|
00000030  20 74 65 78 74 2e 0a 54  68 69 73 20 69 73 20 61  |text..This is a|
00000040  20 32 6e 64 20 6c 69 6e  65 20 6f 66 20 74 65 78  |2nd line of tex|
00000050  74 2e 0a 54 68 69 73 20  69 73 20 61 20 74 68 69  |t..This is a thi|
00000060  72 64 20 6c 69 6e 65 20  6f 66 20 74 65 78 74 2e  |rd line of text.|
00000070  0a 54 68 69 73 20 69 73  20 61 20 6c 69 6e 65 20  |.This is a line |
00000080  6f 66 20 6e 75 6d 62 65  72 73 3a 20 31 32 33 34  |of numbers: 1234|
00000090  35 36 37 38 39 30 0a 54  68 69 73 20 69 73 20 61  |567890.This is a|
000000a0  20 6c 69 6e 65 20 6f 66  20 6e 6f 6e 61 6c 70 68  |line of nonalph|
000000b0  61 20 63 68 61 72 61 63  74 65 72 73 3a 20 21 40  |a characters: !@|
000000c0  23 24 25 5e 26 2a 28 29  2d 2b 0a 0a                |#$%^&*()-+..|
000000cc

```

The **wipeslack** option removes the hidden text from the file.

```

root@kali:~/hexdump# bmap --mode wipeslack textfile-2.txt --verbose
stuffing block 40915
file size was: 204
slack size: 3892
block size: 4096
write error
write error
write error
root@kali:~/hexdump# bmap --mode slack textfile-2.txt --verbose
getting from block 40915
file size was: 204
slack size: 3892
block size: 4096

```

Tools

Windows:

Older versions of Windows had a hex editing command called **debug** built in. PowerShell versions 5.0, 5.1, and 6 have a cmdlet called **Format-hex** that can produce hex dumps. If those options are not available, here are some other tools that can be used on Windows:

- **WinDbg**: “Getting Started with WinDbg (User-Mode),” <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-started-with-windbg>. Note: this tool is meant to be used to debug C/C++/C#/JavaScript code, script files, or even executables.
- **WinHex Hex Editor**: <http://www.winhex.com/winhex/hex-editor.html> (shown above).
- **Wireshark** can import and understand a hex dump in the form `od -Ax -tx1 -v` https://www.wireshark.org/docs/wsug_html_chunked/ChIOImportSection.html.

Linux:

- **file**¹⁵ – is a command that does three tests on a file to try to figure out what kind of data is in the file.
- **xxd**¹⁶ – is a command used to create a hex dump of a file or standard input.

¹⁵ http://www.linfo.org/file_command.html

¹⁶ <https://www.systutorials.com/docs/linux/man/1-xxd/>
<https://www.poftut.com/use-linux-xxd-command-tutorial-hex-binary-operations-example/>

- **od**¹⁷ – is a command used to create an octal or hex dump of a file or standard input.
- **hd**¹⁸ – is a command used to create a hex dump of a file or standard input.

Magic Number Analysis Tools

- **file**¹⁹ – is a command that does three tests on a file to try to figure out what kind of data is in the file.
- **Binwalk**²⁰ – is a tool used to analyze, reverse engineer, and extract firmware images.
- **Scalpel**²¹ – is an open-source data-carving tool.
- **Foremost**²² – is a program that does "data carving"; it recovers files based on their headers, footers, and internal data structures.

Magic Number Lists

- Gary Kessler, "File Signatures Table," https://www.garykessler.net/library/file_sigs.html.
- Wikipedia, "List of file signatures," https://en.wikipedia.org/wiki/List_of_file_signatures.
- Also see: BSD Library Functions Manual, libmagic(3) - Magic number recognition library, <http://man7.org/linux/man-pages/man3/libmagic.3.html>.
- Also see: Linux man page, magic(5) – the file command's magic pattern file, <https://linux.die.net/man/5/magic>.

¹⁷ <https://linux.die.net/man/1/od>

¹⁸ <https://linoxide.com/linux-how-to/linux-hexdump-command-examples/>

¹⁹ http://www.linfo.org/file_command.html

²⁰ <https://github.com/ReFirmLabs/binwalk/wiki>

²¹ <https://github.com/sleuthkit/scalpel>

²² <http://foremost.sourceforge.net/>

References

"ASCII Table and Description," <https://www.asciitable.com/>

Chuvakin, Anton, Ph.D., "Linux Data Hiding and Recovery,"
<http://target0.be/madchat/crypto/stegano/unix/covert/data-hiding-forensics.html>

Crowley, Dan, "Jack of All Formats,"
<https://www.slideshare.net/SOURCEConference/dan-crowley-jack-of-all-formats>

Davis, Jeremy, MacLean, Joe, Dampier, David, "Methods of Information Hiding and Detection in File Systems,"
https://www.researchgate.net/profile/David_Dampier/publication/221411287_Methods_of_Information_Hiding_and_Detection_in_File_Systems/links/5581b8b508ae6cf036c16d26/Methods-of-Information-Hiding-and-Detection-in-File-Systems.pdf

Hackerkitty, "Installing bmap in Linux," <https://hackerkitty.wordpress.com/2014/10/20/installing-bmap-in-linux/>

InfoSec Institute, "Steganography: What your eyes don't see,"
<http://resources.infosecinstitute.com/steganography-what-your-eyes-dont-see/>

Irongeek.com, "Practical guide to alternative data streams in NTFS,"
<https://www.irongeek.com/i.php?page=security/altds>

Microsoft Technet, Jeff Hughes, "NTFS File Attributes,"
<https://blogs.technet.microsoft.com/askcore/2010/08/25/ntfs-file-attributes/>

Microsoft Technet, Hey, Scripting Guy! Blog, "Playing Around with Format-Hex,"
<https://blogs.technet.microsoft.com/heyscriptingguy/2015/08/06/playing-around-with-format-hex/>

Mitchell, Robert, Senior Support Escalation Engineer, Microsoft Technet, Alternate Data Streams

- "NTFS file attributes,"
<https://blogs.technet.microsoft.com/askcore/2010/08/25/ntfs-file-attributes/>
- "Alternate data streams in NTFS,"
<https://blogs.technet.microsoft.com/askcore/2013/03/24/alternate-data-streams-in-ntfs/>

Olensky, Sven, https://digital-forensics.sans.org/community/papers/gcfa/windows-shellbag-forensics-in-depth_80

Rootkit Analytics, "Exploring Alternate Data Streams,"
<http://www.rootkitanalytics.com/userland/Exploring-Alternate-Data-Streams.php>

Security Box, "Slack Space Hiding," <https://security-box.org/article35/slack-space-hiding>

Sivathasan, P. M., "Steganography: An Urban Myth Or A Virtual Reality,"
https://minerva.leeds.ac.uk/bbcswebdav/orgs/SCH_Computing/FYProj/reports/0102/sivathasan.pdf

StackExchange, "How to interpret an octal or hex dump of a binary file?"
<https://unix.stackexchange.com/questions/33673/how-to-interpret-an-octal-or-hex-dump-of-a-binary-file>

Wikipedia, "Data Cluster," https://en.wikipedia.org/wiki/Data_cluster

Wireshark.org, "5.5. Import hex dump,"
https://www.wireshark.org/docs/wsug_html_chunked/ChIOImportSection.html

WonderHowTo, Null-bytes, "How to Hide Secret Data Inside an Image or Audio File in Seconds,"
<https://null-byte.wonderhowto.com/how-to/steganography-hide-secret-data-inside-image-audio-file-seconds-0180936/>