virus BULLETIN Covering the global threat landscape

# SCRIPT IN A LOSSY STREAM

*Dénes Óvári*
CSIS, Denmark

Some years ago, developers of exploit kits began to use malformed PDF files as attack vectors for malicious drive-by downloads, usually by exploiting vulnerabilities present in viewer applications. Detections were duly added to AV products and as a result, the generated PDF files became increasingly obfuscated as malware attempted to circumvent the scanners.

Typically, advantage was taken of the wide range of filters that are provided by the PDF specification for streams in a document. Besides the various text encodings and common data compressors such as Deflate and LZW, even image compressors such as CCITTFaxDecode [1] and JBIG2Decode [2] were seen storing payloads in the wild – all due to the fact that a binary stream can usually be interpreted as raw image data.

## LOSSY IMAGE COMPRESSORS IN PDF

Consequently, scanners were upgraded to handle the compression of streams in PDF files. However, mainly for performance reasons, certain assumptions had to be made about the filters. For example, streams compressed using lossy compressors like JPXDecode and DCTDecode (which is a JPEG-compatible filter) are skipped by scanners and even by popular PDF forensics tools – indicating that their use for the storage of malicious payloads has been deemed impossible by their developers.

In a presentation about PDF heuristics [3], the presence of DCTDecode streams in a document was said to be indicative of a clean file. This was with good reason, of course – the PDF specification states that the uncompressed data would only be an approximation of the original data [4], implying exclusive use of the filter for photograph-like images.

## DELVING INTO JPEG

But is that really the case? JPEG compression has a couple of options, and likewise the PDF specification contains a handful of ways to determine how the decompressed data should be interpreted.

Briefly: after optionally downsampling some of the components, the baseline JPEG splits the image data into 8x8 pixel blocks (MCUs). The original unsigned values are scaled to signed values by subtracting 128 (see MCU number 1 with sample image data in Figure 1). Afterwards, the data is transformed to the frequency domain by means of a two-dimensional discrete cosine transform (see Figure 1).

The results from DCT (called coefficients) are divided with a quality factor ($q_f$) dependent quantization matrix (see Figure 2). Rounding of the results leads to the dropping of certain high-frequency components of the image (see Figure 2) – and that is the stage at which most of the data loss occurs. Finally, all of the processed data is losslessly compressed using a form of Huffman coding.

$$\begin{bmatrix} 27 & 26 & 41 & 65 & 66 & 39 & 34 & 17 \\ 26 & 29 & 38 & 47 & 28 & 23 & 12 & 12 \\ 41 & 38 & 47 & 28 & 23 & 12 & 12 & 12 \\ 65 & 47 & 28 & 23 & 12 & 12 & 12 & 12 \\ 66 & 28 & 23 & 12 & 12 & 12 & 12 & 12 \\ 39 & 23 & 12 & 12 & 12 & 12 & 12 & 12 \\ 34 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 17 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \end{bmatrix} \quad \begin{bmatrix} -15 & -1 & -1 & 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & -2 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 2 & -1 & -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3. 4.

*Figure 2: The coefficients are divided with a quality factor dependent quantization matrix (3); rounding of the results leads to the dropping of certain high-frequency components of the image (4).*

Decompression is performed backwards: in a nutshell, the data is multiplied with the quantization table, an inverse DCT is performed, and the values are shifted back to the 0–255 range.

At high $q_f$ settings, with floating-point precision DCT calculation, it would be possible to store and retrieve raw RGB data losslessly, using software like GIMP, for example. However, JPEG implementations differ – quantization tables

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \quad \begin{bmatrix} -415.3750 & -30.1857 & -61.1971 & 27.2393 & 56.1250 & -20.0952 & -2.3876 & 0.4618 \\ 4.4655 & -21.8574 & -60.7580 & 10.2536 & 13.1451 & -7.0874 & -8.5354 & 4.8769 \\ -46.8345 & 7.3706 & 77.1294 & -24.5620 & -28.9117 & 9.9335 & 5.4168 & -5.6490 \\ -48.5350 & 12.0684 & 34.0998 & -14.7594 & -10.2406 & 6.2960 & 1.8312 & 1.9459 \\ 12.1250 & -6.5534 & -13.1961 & -3.9514 & -1.8750 & 1.7453 & -2.7872 & 3.1353 \\ -7.7347 & 2.9055 & 2.3798 & -5.9393 & -2.3778 & 0.9414 & 4.3037 & 1.8487 \\ -1.0307 & 0.1831 & 0.4168 & -2.4156 & -0.8778 & -3.0193 & 4.1206 & -0.6619 \\ -0.1654 & 0.1416 & -1.0715 & -4.1929 & -1.1703 & -0.0978 & 0.5013 & 1.6755 \end{bmatrix}$$

1. 2.

*Figure 1: (1) MCU with sample image data; (2) data is transformed by means of a two-dimensional discrete cosine transform.*

and certain stages of decompression are entirely up to the developer, therefore the output might be different when the stream is decompressed with another library.

In the most popular PDF reader application, *Acrobat Reader*, we can see that *Adobe*'s JPEG implementation could alter some samples in the LSB +/- 1 range [5]. This is completely reasonable for image reproduction and conforms to the JPEG specification, while making the misuse of DCTDecode to store arbitrary data also impossible at first sight.

## COLOUR SPACE CONVERSION

However, only the colour mode of JPEG has been inspected so far, where the image is actually stored in the YCbCr colour space, using certain properties of the human visual system to increase efficiency. Practically, this means converting RGB values into luminance (Y) and two chrominance (Cb, Cr) components with a set of equations [6] before encoding.

If these calculations are computed with finite precision, rounding errors could occur, causing information loss – certain RGB values are impossible to represent in the output. Since at high $q_f$ settings, the quantization tables contain only 1s, it could be assumed that actually *all* of the information loss was due to this conversion.

This assumption can be verified because JPEG has a separate greyscale mode. Omitting any colour space conversion, using only the luminance layer, every 24 bits of incoming data represent only a single pixel of the image.

## PROOF OF CONCEPT

A PoC file was made, where a short script was encoded as a greyscale JPEG image with the highest quality setting. The script was padded with 0x00 bytes until the next MCU

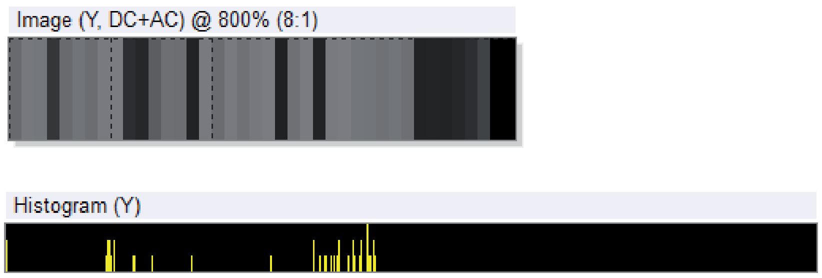
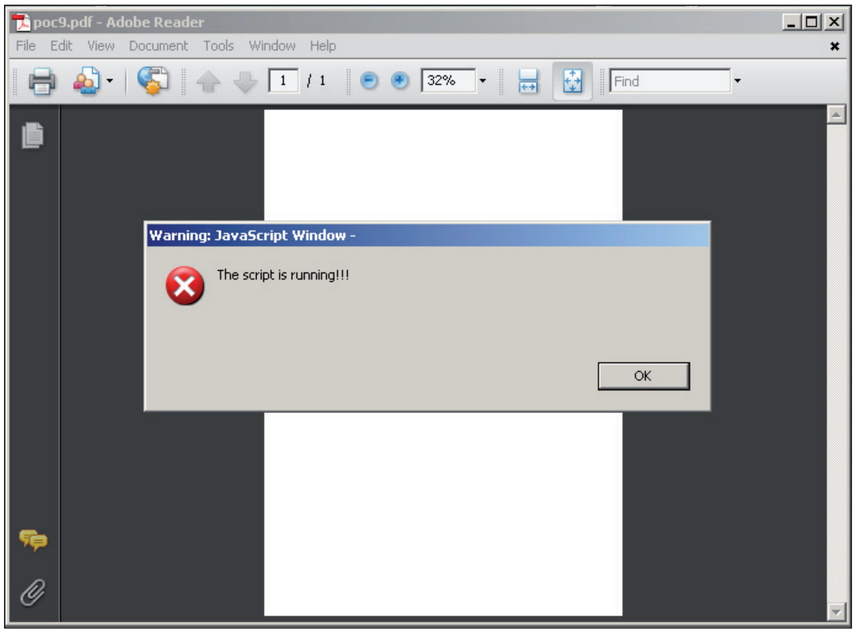
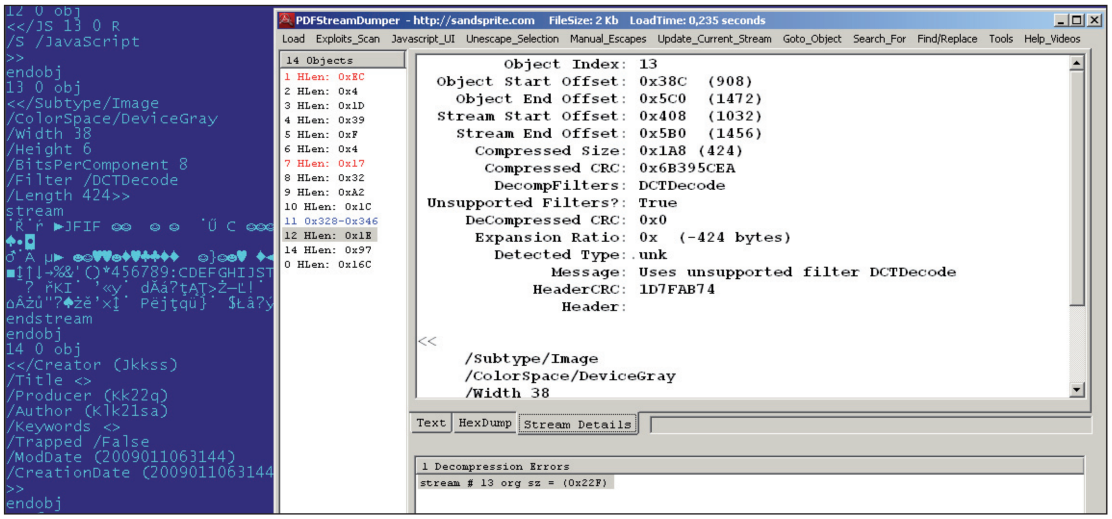


*Figure 3: The script used for demonstration, encoded as a greyscale image.*


poc9.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
32%
Find
Warning: JavaScript Window -
OK

The script is running!!!

*Figure 4: After opening the file in Reader 9.*


PDFStreamDumper - http://sandsprite.com FileSize: 2 Kb LoadTime: 0,235 seconds
Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos
Text HexDump Stream Details

```
12 0 obj
<</JS 13 0 R
/S /JavaScript
>>
endobj
13 0 obj
<</Subtype/Image
/ColorSpace/DeviceGray
/Width 38
/Height 6
/BitsPerComponent 8
/Filter /DCTDecode
/Length 424>>
stream
˙Ř˙ŕ ►JFIF ☻☺  ☺ ☺  ˙Ű C ☻☻☻
♠•◘
♂˙A µ► ☻☻♥♥☻♦♥♣♣♦♦  ☺}☻☻♥ ♦◄
■↨↑↓→%&'()*456789:CDEFGHIJST
 ? řKI  '«y˙ dĂá?ţĄŢ>Ż—Ľ!˙
⌂ÂżůŽ"?♠žě'x↨˙ Pějţqü}˙ $Łâ?ý
endstream
endobj
14 0 obj
<</Creator (Jkkss)
/Title <>
/Producer (Kk22q)
/Author (Klk21sa)
/Keywords <>
/Trapped /False
/ModDate (2009011063144)
/CreationDate (2009011063144
>>
endobj
```

```
14 Objects
1 HLen: 0xEC
2 HLen: 0x4
3 HLen: 0x1D
4 HLen: 0x39
5 HLen: 0xF
6 HLen: 0x4
7 HLen: 0x17
8 HLen: 0x32
9 HLen: 0xA2
10 HLen: 0x1C
11 0x328-0x346
12 HLen: 0x1E
14 HLen: 0x97
0 HLen: 0x16C
```

```
        Object Index:  13
 Object Start Offset:  0x38C  (908)
   Object End Offset:  0x5C0  (1472)
 Stream Start Offset:  0x408  (1032)
   Stream End Offset:  0x5B0  (1456)
     Compressed Size:  0x1A8 (424)
      Compressed CRC:  0x6B395CEA
        DecompFilters: DCTDecode
Unsupported Filters?:  True
    DeCompressed CRC:  0x0
     Expansion Ratio:  0x  (-424 bytes)
       Detected Type: .unk
             Message:  Uses unsupported filter DCTDecode
           HeaderCRC:  1D7FAB74
              Header:

<<
     /Subtype/Image
     /ColorSpace/DeviceGray
     /Width 38
```

```
1 Decompression Errors
stream # 13 org sz = (0x22F)
```

*Figure 5: The script is invisible in PdfStreamDumper.*

boundary, and this was repeated seven more times. Then it was placed in an Image object filtered with DCTDecode, which was referenced by a JavaScript action entry.

When opening the document, the alert dialog just pops up under the old *Reader 9* (Figure 4), proving that the code of the short script was decompressed losslessly.

Under *Reader XI*, certain bits changed in the decompressed data, rendering the original file unusable. However, simply changing a couple of characters in each MCU of the stream until the decompressed data looked as it was expected to was enough for the file to work again.

## CONCLUSIONS

Following the introduction of a sandbox for JavaScript code in *Acrobat Reader*, the use of PDF as an attack vector decreased dramatically. However, the PoC file described here demonstrates a new way to store data in PDF files. Although this is not a security breach in itself (an exploit still needs to be used inside the stream for malicious activity), the fact that the usage of DCTDecode for this purpose has seemingly been ruled out by the industry means that even known threats could be hidden in this way from anti-virus scanners and/or researchers.

In order to provide users with maximum protection, the DCTDecode stream must no longer be overlooked: in PDF reader implementations, the referencing of uncompressed image data as parameters from objects expecting binary data should be prohibited. We should also perhaps re-examine the handling of other file formats in which data in JPEG format is assumed always to be lossily compressed, while a greyscale mode is still available.

## REFERENCES

[1] Baccas, P. PDF malware adopts another obfuscation trick in attempt to avoid detection. http://nakedsecurity.sophos.com/2012/04/05/ccittfax-pdf-malware/.

[2] Sejtko, J. Another nasty trick in malicious PDF. http://blog.avast.com/2011/04/22/another-nasty-trick-in-malicious-pdf/.

[3] Baccas, P. Malicious PDFs: A summary of my VB2010 presentation. http://nakedsecurity.sophos.com/2010/10/08/malicious-pdfs-points-vb2010-presentation/.

[4] PDF Reference, version 1.7, table 3.5 'Standard filters'.

[5] Supporting the DCT Filters in PostScript Level 2. Technical Note #5116, Adobe Systems Incorporated. Section 23.

[6] Hamilton, E. JPEG File Interchange Format, version 1.02. http://www.w3.org/Graphics/JPEG/jfif3.pdf (p.3).

**Editor:** Martijn Grooten
**Chief of Operations:** John Hawes
**Security Test Engineers:** Scott James, Tony Oliveira, Adrian Luca
**Sales Executive:** Allison Sketchley
**Editorial Assistant:** Helen Martin
**Consultant Technical Editors:** Dr Morton Swimmer, Ian Whalley

**© 2015 Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England.**
**Tel: +44 (0)1235 555139. Fax: +44 (0)1865 543153**
**Email: editorial@virusbtn.com**
**Web: http://www.virusbtn.com/**