



# An Examination of the Redaction Functionality of Adobe Acrobat Pro DC 2017

**First published:** July 2018  
**Last updated:** October 2021

## Introduction

There have been numerous cases of security breaches resulting from a failure to effectively redact sensitive or private information from documents prior to release into the public domain. To assist in mitigating this security risk, Adobe Acrobat Pro DC 2017 provides redaction and sanitisation functionality that aims to completely remove undesirable information and other hidden information (e.g. metadata) from PDF documents.

This publication provides guidance on the efficacy of redaction facilities within Adobe Acrobat Pro DC 2017 and is intended for information technology and information security professionals within organisations looking to redact sensitive or personal information from PDF documents before releasing them into the public domain or to other third parties.

## Scope of testing

The redaction functionality in Adobe Acrobat Pro 10 was previously tested in 2011 to determine if any redacted information could be recovered from PDF documents. The current round of testing aimed to examine the same functionality previously tested but in Adobe Acrobat Pro DC 2017.

For the purposes of this publication, the definition of successful redaction was the complete removal of redacted data from every location in a PDF document's file structure.

As part of testing, a number of test cases were considered that represented some of the different ways that information could be stored within a PDF document. This included:

- embedded text
- embedded image
- data from historical editing
- interactive form
- embedded text obscured by an embedded image
- embedded text in an encrypted PDF
- embedded metadata.

Unless otherwise stated, the following application versions were used:

- Adobe Acrobat Pro DC 2017 (2017.012.20093) which installs Adobe PDF Library 15 and Adobe Acrobat Distiller 17
- Microsoft Word 2010 (14.0.6023.1)

- LibreOffice Writer 5.1.6.2
- CutePDF Writer 3.2 which installs Ghostscript 8.15.

The Calibri font was used in Microsoft Office documents in Microsoft Windows. This font was installed in Ubuntu Linux so that test files could be opened in LibreOffice Writer.

PDF documents were generated using each of the below rendering engines:

- Adobe Acrobat (Using the 'Create PDF' Microsoft Word add-in or native PDF authoring within Adobe Acrobat, both of which use the Adobe PDF Library)
- Adobe Acrobat Distiller (Printing to the Adobe PDF printer)
- Microsoft Word (Using 'Save As' PDF functionality)
- CutePDF (Printing to the CutePDF Writer printer)
- LibreOffice Writer (Using the 'Export As PDF' functionality).

In some cases, not all rendering engines were tested as not all possessed the necessary functionality.

For the purposes of testing, the application used to create the PDF documents refers to the rendering engine that did the PDF conversion. For example, using the 'Create PDF' Microsoft Word add-in installed by Adobe Acrobat, the PDF conversion is performed by the Adobe Acrobat rendering engine (Adobe PDF Library). Similarly, when printing to the Adobe PDF printer installed by Adobe Acrobat within Microsoft Word, the file conversion is performed by the Adobe Acrobat Distiller rendering engine. Only choosing to save the file by selecting the 'Save As' PDF option in Microsoft Word results in a PDF being rendered by Microsoft Word.

The previous testing conducted in 2011 used a single rendering engine to create PDF documents. In the current round of testing, PDF documents using different rendering engines were used to determine whether the source of the PDF document had any impact upon successful redaction.

Depending on the rendering engines used, PDF documents were generated using different versions of the PDF standard:

- Adobe Acrobat
  - Adobe PDF Library (PDF version 1.5)
  - Adobe Acrobat Distiller (PDF version 1.5)
- Microsoft Word (PDF version 1.5)
- Cute PDF (PDF version 1.4)
- LibreOffice Writer (PDF version 1.4).

Using the functions of Adobe Acrobat, PDF documents were generated using different versions of the PDF standard:

- interactive form (PDF version 1.6)
- encryption (PDF version 1.6)
- sanitisation (PDF version 1.6)
- redaction (PDF version 1.7).

The redacted PDF documents were analysed with free or open source tools to determine whether any redacted information could be recovered:

- [Pdfminer toolkit \(pdf2txt\)](#)
- [Poppler toolkit \(pdfimages\)](#)
- [Origami toolkit \(pdfwalker\)](#)
- [PDF Stream Dumper 9.3.](#)

# Testing results and recommendations

## Successful redaction outcomes

No redacted information was recovered from PDF documents created with Adobe Acrobat, Adobe Acrobat Distiller and Microsoft Word. This result is similar to that of previous testing conducted in 2011 which examined the redaction functionality of Adobe Acrobat Pro 10.

## Failures in redacting information

Remnants of redacted information were recovered from PDF documents created with CutePDF and LibreOffice Writer.

The remnants of redacted information were located within [objects containing embedded font maps](#) (CMap objects). The ability to recover these data remnants was the result of differences in the mechanisms used by CutePDF and LibreOffice Writer to embed font maps, and the Adobe Acrobat redaction functionality's inability to identify and remove them. The Adobe Acrobat sanitisation functionality also failed to remove these data remnants.

It is not known whether PDF documents created by rendering engines not tested during this round of testing will also fail to be successfully redacted. Until this is known, assurance that data remnants cannot be recovered from redacted PDF documents requires that the creation of PDF documents be restricted to Adobe Acrobat, Adobe Acrobat Distiller and Microsoft Word.

To assist in identifying the software used to create a PDF document, the metadata can be examined via the document's properties in Adobe Acrobat or Adobe Acrobat Reader. If the PDF document was created with Adobe Acrobat, Adobe Acrobat Distiller or Microsoft Word, the PDF Producer field will contain 'Adobe PDF Library', 'Acrobat Distiller' or 'Microsoft Word' respectively. If the PDF Producer field contains something else, there is a chance that redaction of sensitive or private information might fail. Note that if the PDF document had been previously sanitised, the metadata would have been deleted and the PDF Producer field will be empty. In these cases it should be treated as if it was created by a rendering engine that cannot be successfully redacted by Adobe Acrobat.

## Detailed testing results

For a full breakdown and discussion of the testing results see Appendix A and B respectively.

## Recommendations

When a requirement exists to redact sensitive or private information from PDF documents before releasing them into the public domain or to other third parties, organisations should:

- verify the original PDF document was created using Adobe Acrobat, Adobe Acrobat Distiller or Microsoft Word by checking the metadata of the PDF document
- perform redaction and sanitisation of the document using Adobe Acrobat Pro DC 2017.

## Further information

The [Information Security Manual](#) is a cyber security framework that organisations can apply to protect their systems and data from cyber threats. The advice in the [Strategies to Mitigate Cyber Security Incidents](#), along with its [Essential Eight](#), complements this framework.

A guide to [redacting sensitive information from PDF documents](#), including step-by-step instructions, is available from Adobe.

## Contact details

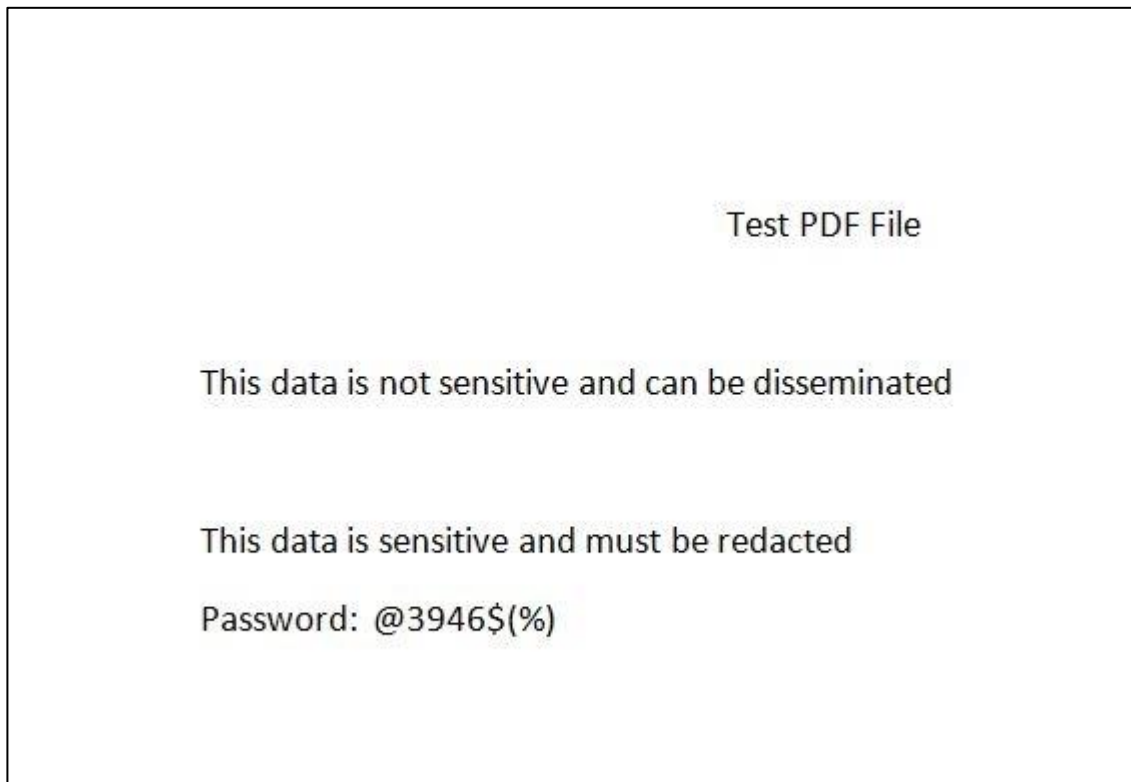
If you have any questions regarding this guidance you can [write to us](#) or call us on 1300 CYBER1 (1300 292 371).

# Appendix A: Detailed testing results

## Test 1: Redaction of embedded text

The aim of Test 1 was to determine whether remnants of redacted text could be found in PDF documents redacted with Adobe Acrobat Pro DC 2017.

A Microsoft Word document was created that contained a title and three lines of text. The last two lines represented sensitive information.



A corresponding PDF document was created using each of the rendering engines being examined: Adobe Acrobat, Adobe Acrobat Distiller, Microsoft Word, CutePDF and LibreOffice Writer. This represented five PDF documents.

The internal structures of the PDF documents were parsed with the PDF Stream Dumper tool. For each of the PDF documents, the objects within the file structures that contained embedded text were identified. For example, the embedded text object from the file generated using Adobe Acrobat is shown below with embedded text highlighted in green.

```
BT
/P <</MCID 0 >>BDC
/CS0 cs 0 scn
/TT0 1 Tf
0.002 Tc -0.006 Tw 14.04 -0 0 14.04 269.76 706.56 Tm
[(T)1.8 (e)4.3 (st)3.7 ( P)6.2 (D)1.6 (F)-0.6 ( File)]TJ
0 Tc 0 Tw 5.154 0 Td
( )Tj
EMC
/P <</MCID 1 >>BDC
-2.573 -2.111 Td
( )Tj
EMC
/P <</MCID 2 >>BDC
-0.004 Tc -16.667 -2.111 Td
[(T)-4.2 (hi)-5.7 (s)-6.2 ( da)-3.6 (t)-2.3 (a)-3.6 ( i)-5.7 (s)-6.2 (
no)-6.9 (t)-2.3 ( a)-6.2 (e)-1.7 (na)-6.1 (i)-5.8 (t)-2.3 (i)-5.8 (v)-5
(e)-1.7 ( a)-3.7 (nd ca)-3.7 (n)-8.9 ( be)-1.7 ( )-8.8 (di)-5.8 (a)-6.1
(s)-6.2 (e)-1.7 (mi)-5.8 (na)-3.7 (t)-2.3 (e)-1.7 (d)]TJ
0 Tc 20.171 0 Td
( )Tj
EMC
/P <</MCID 3 >>BDC
-20.171 -2.111 Td
( )Tj
EMC
/P <</MCID 4 >>BDC
-0.004 Tc 0 -2.111 Td
[(T)-4.2 (hi)-5.7 (s)-6.2 ( da)-3.6 (t)-2.3 (a)-3.6 ( i)-5.7 (s)-6.2 (
s)-6.2 (e)-1.7 (na)-6.1 (i)-5.8 (t)-2.3 (i)-5.8 (v)-5 (e)-1.7 ( a)-3.7
(nd mus)-6.2 (t)-2.3 ( )-8.8 (be)-1.7 ( r)-5.4 (e)-1.8 (da)-3.7 (c)-8.3
(t)-2.3 (e)-10.3 (d)]TJ
0 Tc ( )Tj
EMC
/P <</MCID 5 >>BDC
-0.003 Tc -0.001 Tw 0 -2.12 Td
[(P)1.2 (a)-2.7 (s)-5.1 (s)-5.2 (w)2.6 (o)-5.9 (r)-4.5 (d)0.7 ( : e)-
6.4 {394}]TJ
0.003 Tw [(6$ \ ( )-7.7 (%)-6 (\ ))]TJ
0 Tc 0 Tw 11.04 -0 0 11.04 203.16 558.24 Tm
( )Tj
EMC
ET
```

Examination of the embedded structural objects within each PDF document revealed that all rendering engines utilised font subsets to reduce file size. However, in regard to the mapping of character codes to character selectors (glyphs), different engines used alternate mechanisms.

Microsoft Word did not embed a CMap but instead used pre-defined WinAnsiEncoding. Adobe Acrobat Distiller only embedded a CMap for a single character mapping and for the remaining characters utilised pre-defined WinAnsiEncoding.

Adobe Acrobat embedded a custom ToUnicode CMap within the PDF document. The order of the mappings in the CMap reflected that in Unicode.

```
/CIDInit /ProcSet findresource begin
12 dict begin
begincmap
/CIDSystemInfo
<< /Registry (Adobe)
/Ordering (UCS) /Supplement 0 >> def
/CMapName /Adobe-Identity-UCS def
/CMapType 2 def
1 begincodespacerange
<0000> <FFFF>
endcodespacerange
32 beginbfchar
<20> <0020>
<24> <0024>
<25> <0025>
<28> <0028>
<29> <0029>
<33> <0033>
<34> <0034>
<36> <0036>
<39> <0039>
<3A> <003A>
<40> <0040>
<44> <0044>
<46> <0046>
<50> <0050>
<54> <0054>
<61> <0061>
<62> <0062>
<63> <0063>
<64> <0064>
<65> <0065>
<68> <0068>
<69> <0069>
<6C> <006C>
<6D> <006D>
<6E> <006E>
<6F> <006F>
<72> <0072>
<73> <0073>
<74> <0074>
<75> <0075>
<76> <0076>
<77> <0077>
endbfchar
endcmap CMapName currentdict /CMap defineresource pop end end
```

Mapping order  
reflects that in  
Unicode Standard

In contrast, the PDF documents created with CutePDF (using open source Ghostscript), or open source LibreOffice Writer, both embedded ToUnicode CMap objects where the order of mappings reflected the order that the characters first appeared in text. The CMap object from the former is shown.

```
/CIDInit /ProcSet findresource begin
12 dict begin
begincmap
/CMType 2 def
/CMName/R13 def
1 begincodespacerange
<00><ff>
endcodespacerange
30 beginbfrange
<01><01><0054>
<02><02><0065>
<03><04><0073>
<05><05><0020>
<06><06><0050>
<07><07><0044>
<08><08><0046>
<09><09><0069>
<0a><0a><006c>
<0b><0b><0068>
<0c><0c><0064>
<0d><0d><0061>
<0e><0f><006e>
<10><10><0076>
<11><11><0063>
<12><12><0062>
<13><13><006d>
<14><14><0075>
<15><15><0072>
<16><16><0077>
<17><17><003a>
<18><18><0040>
<19><19><0033>
<1a><1a><0039>
<1b><1b><0034>
<1c><1c><0036>
<1d><1d><0024>
<1e><1e><0028>
<1f><1f><0025>
<20><20><0029>
endbfrange
endcmap
CMName currentdict /CMName defineresource pop
end end
```

Mapping order  
reflects order that  
characters first appear  
in text

Apart from facilitating the mapping of character codes to character selectors, the CMap created an artefact where the order of the mappings itself encoded a text string. The string contained the password from the PDF document which is highlighted in red below.

Test PDFilhdanvcbmurw:@3946\$(%)

The bottom two lines of text were then redacted from each PDF document with Adobe Acrobat.

Test PDF File

This data is not sensitive and can be disseminated

[REDACTED]

[REDACTED]



The internal structures of the redacted PDF documents were parsed with the PDF Stream Dumper tool. In all cases, the redacted text was successfully removed from the embedded PDF text objects. For example, the object from the PDF document produced by Adobe Acrobat is shown below with embedded text highlighted in green.

```
BT
/P <</MCID 0 >>>BDC
/CS0 cs 0 scn
/TT0 1 Tf
0.002 Tc -0.006 Tw 14.04 0 0 14.04 269.76 706.56 Tm
[(T)1.8(e)4.3(st)3.7( P)6.2(D)1.6(F)-0.6( File)]TJ
0 Tc 0 Tw 5.154 0 Td
( )Tj
EMC
/P <</MCID 1 >>>BDC
-2.573 -2.111 Td
( )Tj
EMC
/P <</MCID 2 >>>BDC
-0.004 Tc -16.667 -2.111 Td
[(T)-4.2(hi)-5.7(s)-6.2( da)-3.6(t)-2.3(a)-3.6( i)-5.7(s)-6.2( no)-6.9
(t)-2.3( a)-6.2(e)-1.7(ns)-6.1(i)-5.8(t)-2.3(i)-5.8(v)-5(e)-1.7( a)-3.7
(nd ca)-3.7(n)-8.9( be)-1.7( )-8.8(di)-5.8(s)-6.1(a)-6.2(e)-1.7(mi)-5.8
(na)-3.7(t)-2.3(e)-1.7(d)]TJ
0 Tc 20.171 0 Td
( )Tj
EMC
/P <</MCID 3 >>>BDC
-20.171 -2.111 Td
( )Tj
EMC
/P <</MCID 4 >>>BDC
17.389 -2.111 Td
( )Tj
EMC
/P <</MCID 5 >>>BDC
11.04 0 0 11.04 203.16 558.24 Tm
( )Tj
EMC
ET
```

However, examination of the CMap objects within the redacted PDF documents created by CutePDF or LibreOffice Writer revealed that remnants of redacted text remained. For example, the CMap object from the redacted PDF document that was generated with LibreOffice Writer is shown below with the data artefact that reveals the password string shown in red.

```
/Supplement 0
>> def
/CMAPName/Adobe-Identity-UCS def
/CMAPType 2 def
1 begincodespacerange
<00> <FF>
endcodespacerange
33 beginbfchar
<01> <0054>
<02> <0065>
<03> <0073>
<04> <0074>
<05> <0020>
<06> <0050>
<07> <0044>
<08> <0046>
<09> <0069>
<0A> <006C>
<0B> <0068>
<0C> <0064>
<0D> <0061>
<0E> <006E>
<0F> <006F>
<10> <00740069>
<11> <0076>
<12> <0063>
<13> <0062>
<14> <006D>
<15> <0075>
<16> <0072>
<17> <0077>
<18> <003A>
<19> <0040>
<1A> <0033>
<1B> <0039>
<1C> <0034>
<1D> <0036>
<1E> <0024>
<1F> <0028>
<20> <0025>
<21> <0029>
endbfchar
endcmap
CMAPName currentdict /CMAP defineresource pop
end
end
```

It appears that the redaction functionality of Adobe Acrobat does not identify artefacts of redacted text in CMap objects when the PDF document was generated by CutePDF or LibreOffice Writer. In this test case, the redacted password was fully recoverable.

## Test 2: Redaction of text within an embedded image

The aim of Test 2 was to verify that an embedded image containing text within a PDF document was edited by the redaction process and not simply obscured.

A representation of the text from the PDF document in Test 1 was copied into an image file. The image file was in turn used to create five separate PDF documents with the rendering engines used in Test 1.

Each PDF document was analysed with the pdf2txt tool which extracts embedded text. In all cases, no extractable text was found. This was the expected result, given that all text was represented within an embedded image. For example, the result of running the pdf2txt tool against the PDF document generated by Adobe Acrobat is shown below. To verify this result, each PDF document was also parsed with the PDF Stream Dumper tool.

```
user@UBUNTU-2: ~/Desktop
user@UBUNTU-2:~/Desktop$ pdf2txt test2_adobePDFlibrary.pdf
user@UBUNTU-2:~/Desktop$
```

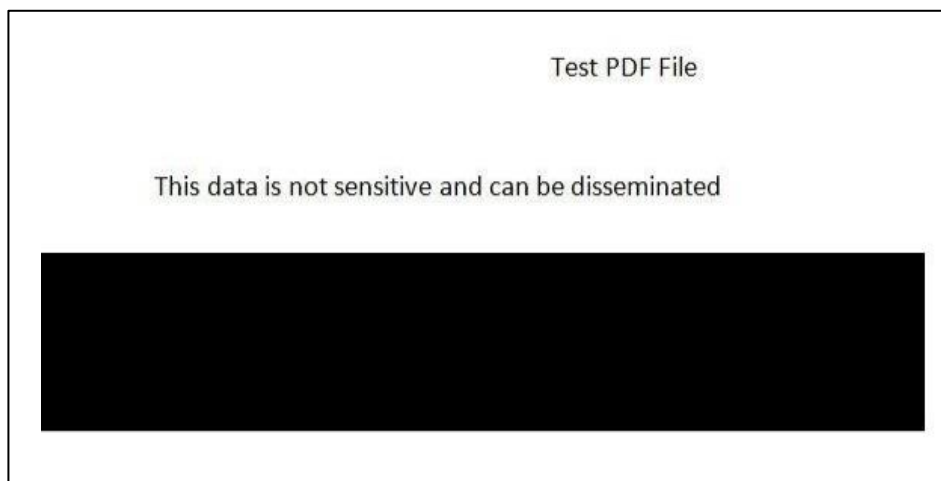
The pdftimages tool, which can detect and analyse embedded images, was run against the PDF documents. The tool successfully identified the embedded images. The output of the tool when run against the PDF document generated by Adobe Acrobat is shown below.

```
user@UBUNTU-2: ~/Desktop
user@UBUNTU-2:~/Desktop$ pdftimages -list test2_adobePDFlibrary.pdf
page  num  type  width height color comp bpc  enc interp  object ID x-ppi y-ppi size ratio
-----
   1    0 image   771   342  icc     3   8  jpeg   no      25  0   124   123 22.9K 3.0%
user@UBUNTU-2:~/Desktop$
```

To test the redaction functionality, the last two lines of the PDF documents (representing sensitive information) were redacted using Adobe Acrobat. The redacted PDF documents were again analysed with the pdftimages tool which revealed that the embedded image file had changed in size. For example, the image file in the redacted PDF document created with Adobe Acrobat had decreased from 22.9Kb to 21.1 Kb indicating it had been edited by the redaction process.

```
user@UBUNTU-2: ~/Desktop
user@UBUNTU-2:~/Desktop$ pdftimages -list test2_adobePDFlibrary_Redacted.pdf
page  num  type  width height color comp bpc  enc interp  object ID x-ppi y-ppi size ratio
-----
   1    0 image   771   342  icc     3   8  image no      31  0   124   123 21.1K 2.7%
user@UBUNTU-2:~/Desktop$
```

To verify that the sensitive information within the embedded image file had been successfully redacted, the embedded image was extracted from the PDF documents using the pdfimages tool and examined. For example, the embedded image file from the PDF document created with Adobe Acrobat is shown below. It had been edited by the redaction process to remove the sensitive information.



### Test 3: Redaction of historical revisions of text

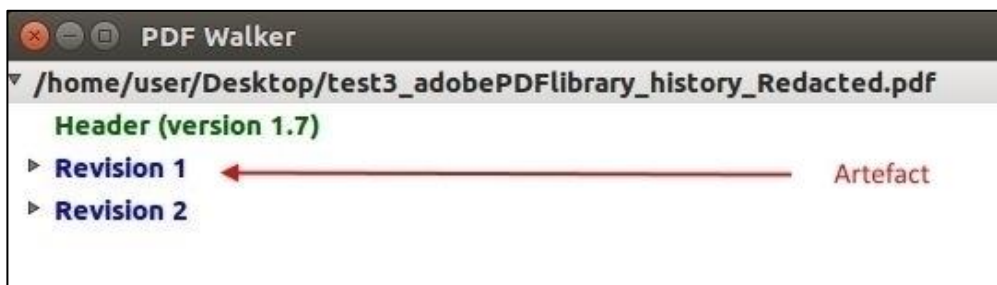
PDF documents store historical revisions of edited text. The aim of Test 3 was to verify that all historical revisions of sensitive text were removed by redaction with Adobe Acrobat.

The PDF documents from Test 1 were opened in Adobe Acrobat. In each case, one of the lines representing sensitive text was edited multiple times, making sure that each edit was saved.

The PDF documents were then parsed with the pdfwalker tool with the revisions of the PDF documents being reflected within the file structures. For example, the file generated by Adobe Acrobat is shown below.



Sensitive text was redacted using Acrobat and the PDF documents were again parsed with the pdfwalker tool. The output of the pdfwalker tool indicated that previous revisions of the sensitive text had been removed. For example, the output from parsing the PDF document generated by Adobe Acrobat is shown below. Note the pdfwalker tool always has 'Revision 1' as an artefact.

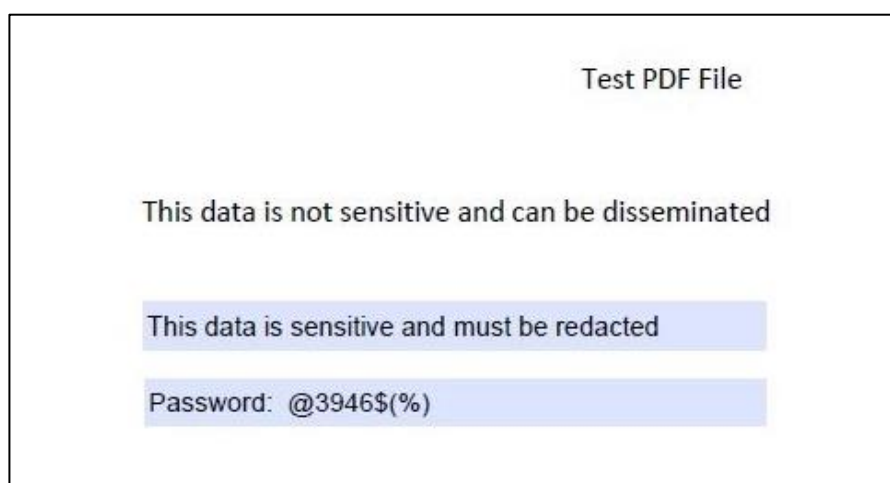


This result was verified by using the PDF Stream Dumper tool to identify the number of file objects in the redacted PDF documents. In the case of the PDF document generated with Adobe Acrobat, the number of file objects had been reduced from 105 to 18, before and after redaction respectively.

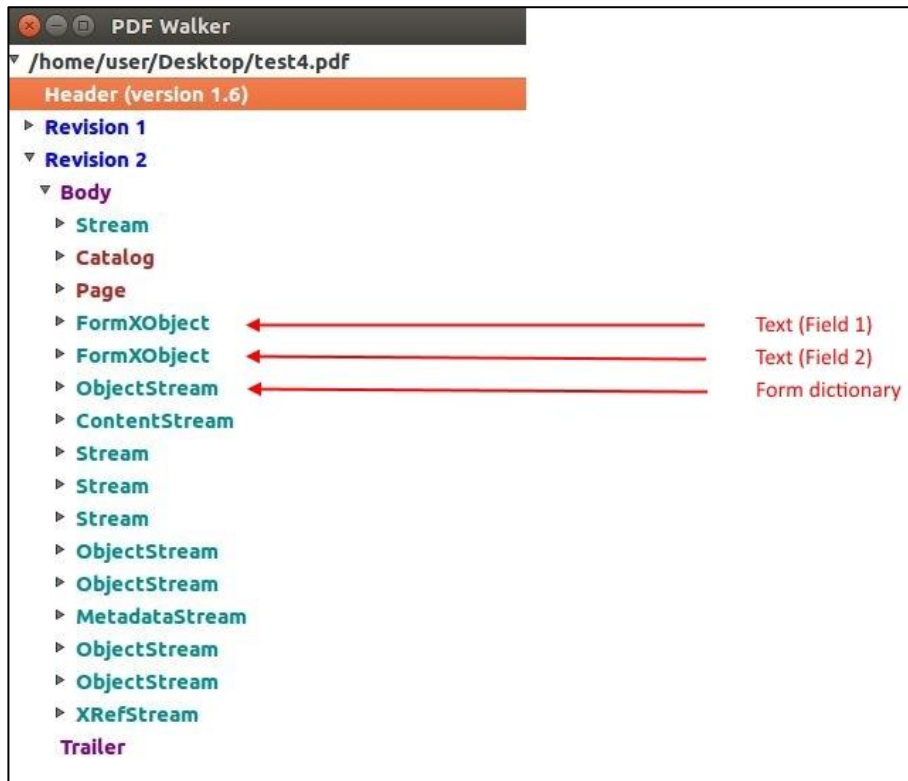
Use of the PDF Stream Dumper tool to examine the embedded text objects showed that in every case historical revisions of sensitive text were removed by redaction. In contrast, the data remnant within CMap objects remained for PDF documents generated with CutePDF or LibreOffice Writer, as was found in Test 1.

## Test 4: Redaction of text within a PDF form

Using Adobe Acrobat, text was entered into two PDF form fields.



The PDF documents were subsequently analysed with the pdfwalker tool. Text was found in three objects shown below. Two of the objects corresponded to each of the two form fields. The third object with text data corresponded to the form dictionary.

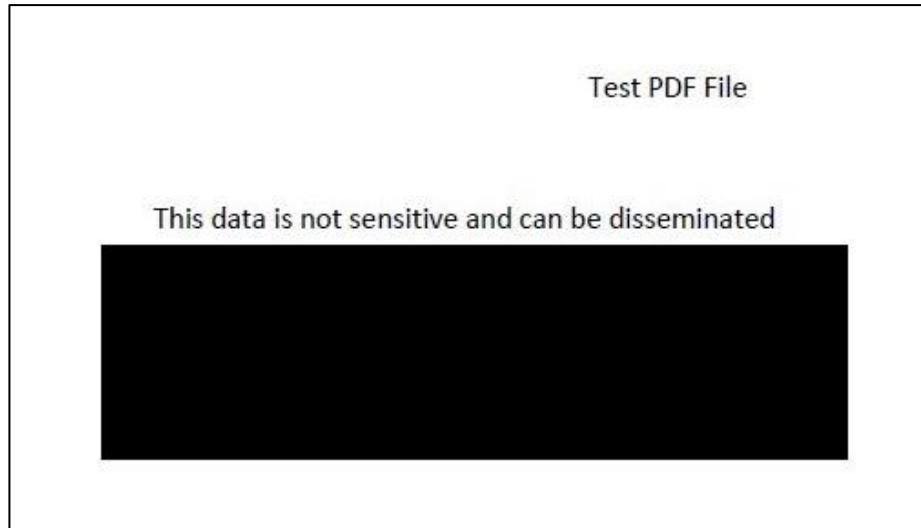


The embedded text within sections of the form dictionary is shown below. Embedded text is highlighted in green.

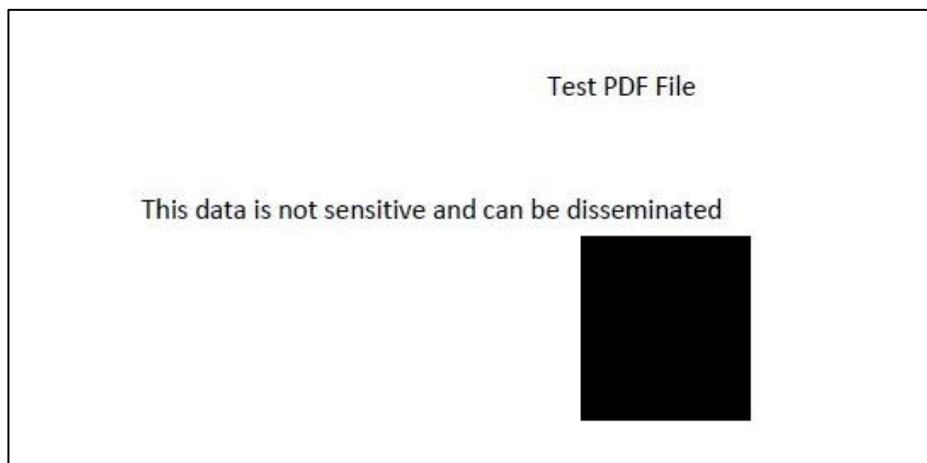
```
30 0 31 120 32 135 33 321 34 394 35 1569 36 1734 37 1752 38 1982 <</DA
(/Helv 0 Tf 0 g )/DR<</Encoding<</PDFDocEncoding 34 0 R>>/Font<</Helv 33
0 R/ZaDb 7 0 R>>>>/Fields[32 0 R 35 0 R]>>>>[32 0 R 35 0 R]<</AP<</N 22 0
R>>/DA(/Helv 12 Tf 0 g)/F 4/FT/Tx/MK<<>>/P 21 0 R/Rect[73.9187 599.594
354.375 627.857]/Subtype/Widget/T(Text1)/Type/Annot/V(This data is
sensitive and must be redacted)>><</BaseFont/Helvetica/Encoding 34 0

593.615]/Subtype/Widget/T(Text2)/Type/Annot/V(Password: @3946$(%))>>
[/ICCBased 26 0 R]<</Ascent 964/CapHeight 632/Descent -307/Flags
32/FontBBox[-503 -307 1240 964]/FontFamily(Calibri)/FontFile2 27 0
R/FontName/VOLHDD+Calibri/FontStretch/Normal/FontWeight 400/ItalicAngle
0/StemV 80/Type/FontDescriptor/XHeight 467>>
<</BaseFont/VOLHDD+Calibri/Encoding/WinAnsiEncoding/FirstChar
32/FontDescriptor 37 0 R/LastChar 118/Subtype/TrueType/ToUnicode 28 0
```

For the first test, both the form fields in the PDF document were fully redacted using Adobe Acrobat.



For the second test, the form fields in the PDF document were partially redacted using Adobe Acrobat.





The redacted PDF documents were then parsed with the pdfwalker tool. In both cases, the form objects were deleted leaving only the object that had contained the form dictionary. The output from parsing the partially redacted PDF document is shown below.



The contents of the remaining form dictionary objects were further analysed with the PDF Stream Dumper tool and no text remnants were found. For example, the content of the form dictionary from the partially redacted PDF document is shown below.

```

31 0 32 105 33 123 34 353 <</DA(/Helv 0 Tf 0 g )/DR<</Encoding
<</PDFDocEncoding 9 0 R>>/Font<</Helv 7 0 R/ZaDb 8 0 R>>>>/Fields[]>>
[/ICCBased 26 0 R]<</Ascent 964/CapHeight 632/Descent -307/Flags
32/FontBBox[-503 -307 1240 964]/FontFamily(Calibri)/FontFile2 27 0
R/FontName/VOLHDD+Calibri/FontStretch/Normal/FontWeight 400/ItalicAngle
0/StemV 80/Type/FontDescriptor/XHeight 467>>
<</BaseFont/VOLHDD+Calibri/Encoding/WinAnsiEncoding/FirstChar
32/FontDescriptor 33 0 R/LastChar 118/Subtype/TrueType/ToUnicode 28 0
R/Type/Font/Widths[226 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 0 0 0 615 0 459 0 0 0 0 0 0 0 0 0 0 0 0 0 517 0 0 0 487 0 0 0 0
0 0 0 0 0 0 479 525 423 525 498 0 0 525 229 0 0 229 799 525 527 0 0 0
391 335 0 452]>>

```

The result was the same for the PDF document where the form fields were fully redacted.

## Test 5: Embedded text obscured with an image

On occasions, attempts at redaction have failed when underlying text was merely obscured by covering it with another layer in the form of a blackened rectangle or image.

Starting with the Microsoft Word file used in Test 1, text was obscured by inserting an overlying image file as shown below. A PDF document was then generated using each of the five rendering engines.



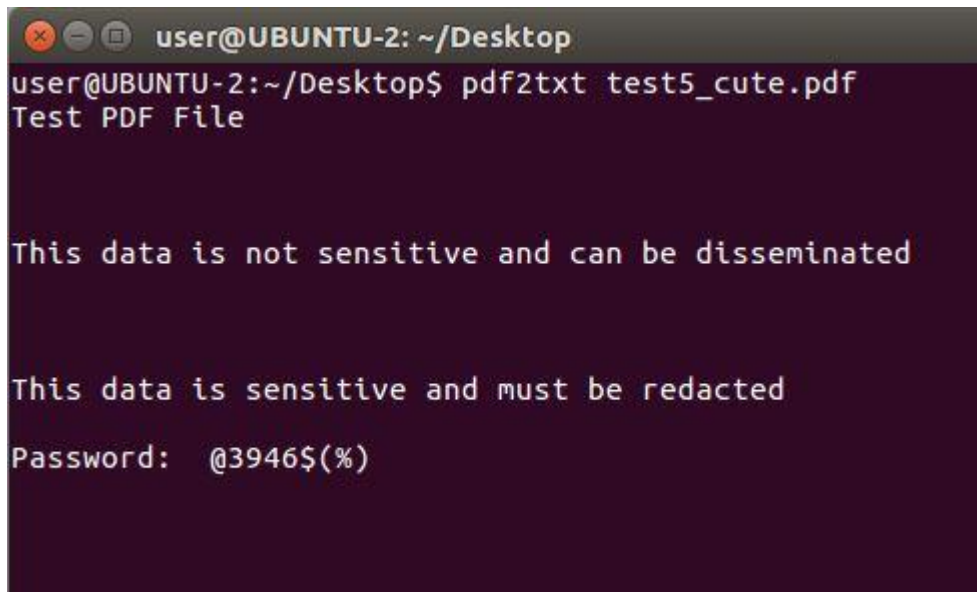
Test·PDF·File¶

¶

This·data·is·not·sensitive·and·can·be·disseminated¶



Using the pdf2txt tool, it was identified that the underlying text remained within the PDF documents despite being obscured with an overlying image. For example, the output of the pdf2txt tool for the PDF document generated with CutePDF is shown below.

A terminal window titled 'user@UBUNTU-2: ~/Desktop' showing the command 'pdf2txt test5\_cute.pdf' and its output. The output consists of three lines of text: 'Test PDF File', 'This data is not sensitive and can be disseminated', and 'This data is sensitive and must be redacted'. Below this, a password prompt 'Password: @3946\$(%)' is shown.

```
user@UBUNTU-2: ~/Desktop
user@UBUNTU-2:~/Desktop$ pdf2txt test5_cute.pdf
Test PDF File

This data is not sensitive and can be disseminated

This data is sensitive and must be redacted

Password: @3946$(%)
```

This demonstrated that obscuring information in a PDF document using an image is not an effective way to redact information.

## Test 6: Redacting encrypted PDF documents

PDF documents from previous tests were encrypted with Adobe Acrobat, redacted and then parsed with PDF analysis tools. The aim of this test was to verify whether encryption changed the underlying file structure of PDF documents or not, and thus if there was any impact on the effectiveness of redaction activities.

For all previous test cases, the results were similar and were unaffected by starting with an encrypted PDF document. For PDF documents created with CutePDF or LibreOffice Writer, as found previously, remnants of redacted text were left in the ToUnicode CMap objects. No other remnants of redacted data were found.

## Test 7: Sanitising PDF documents

Adobe Acrobat offers a sanitisation feature (i.e. 'Remove Hidden Information') that removes hidden data. For example, metadata that might identify the author of a document.

Prior to sanitisation, the PDF documents contained objects with embedded metadata. The PDF documents were each parsed with the PDF Stream Dumper tool and the metadata analysed. The metadata from the PDF document produced by CutePDF is shown below. Useful information is highlighted in green and includes the rendering engine (CutePDF or Ghostscript), the software used to access the rendering engine (Microsoft Word) and the author's user account (IEUser).

```
<?xpacket begin="ï¿" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:meta/" x:xmpk="Adobe XMP Core 5.6-c015
84.159810, 2016/09/10-02:41:30
">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    <rdf:Description rdf:about=""
      xmlns:pdf="http://ns.adobe.com/pdf/1.3/"
      xmlns:xmp="http://ns.adobe.com/xap/1.0/"
      xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
        <pdf:Producer>GEL Ghostscript 9.06</pdf:Producer>
        <xmp:ModifyDate>2017-08-31T17:27:18-07:00</xmp:ModifyDate>
        <xmp:CreateDate>2017-08-30T23:48:57-07:00</xmp:CreateDate>
        <xmp:CreatorTool>PScript5.dll Version 5.2.2</xmp:CreatorTool>
        <xmp:MetadataDate>2017-08-31T17:27:18-07:00</xmp:MetadataDate>
        <xmpMM:DocumentID>uuid:f60eb155-9073-11e7-0000-fd2f4c4ae2ce
      </xmpMM:DocumentID>
        <xmpMM:InstanceID>uuid:3495de6d-a9f4-4372-a262-c8841010aaaf
      </xmpMM:InstanceID>
        <dc:format>application/pdf</dc:format>
        <dc:title>
          <rdf:Alt>
            <rdf:li xml:lang="x-default">Microsoft Word - test1
          </rdf:li>
          </rdf:Alt>
        </dc:title>
        <dc:creator>
          <rdf:Seq>
            <rdf:li>IEUser</rdf:li>
          </rdf:Seq>
        </dc:creator>
      </rdf:Description>
    </rdf:RDF>
  </x:xmpmeta>
```

The metadata is also available from the PDF document's properties within a PDF reader. The metadata from the PDF document produced by CutePDF and read by Adobe Acrobat Reader is shown below.

The screenshot shows the 'Document Properties' dialog box with the 'Description' and 'Advanced' tabs selected. The 'Description' tab contains fields for File, Title, Author, Subject, and Keywords, along with creation and modification dates and the application used. The 'Advanced' tab contains information about the PDF producer, version, location, file size, page size, number of pages, and whether it is a tagged PDF or has fast web view enabled.

Document Properties	
Description   Security   Fonts   Custom   Advanced	
Description	
File:	test7_cute_Redacted.pdf
Title:	Microsoft Word - test1
Author:	IEUser
Subject:	
Keywords:	
Created:	30/08/2017 11:48:57 PM
Modified:	31/08/2017 5:27:18 PM
Application:	PScript5.dll Version 5.2.2
Advanced	
PDF Producer:	GPL Ghostscript 9.06
PDF Version:	1.7 (Acrobat 8.x)
Location:	C:\Users\User\Desktop\PDF\PDF Project (Final)\Analysis\Test 7\
File Size:	19.94 KB (20,415 Bytes)
Page Size:	8.26 x 11.69 in
Number of Pages:	1
Tagged PDF:	No
Fast Web View:	Yes

To check the efficacy of the sanitisation feature, the PDF documents were sanitised and parsed with the pdfwalker tool. In all test cases, sanitisation deleted the object containing metadata. For example, the file structure from the PDF document produced by CutePDF before (left) and after (right) sanitisation is shown below. The object containing metadata is indicated.

<div>/home/user/Documents/test7_cute_Redacted.pdf</div>	<div>/home/user/Documents/test7_cute_Redacted_san.pdf</div>
<div>Header (version 1.7)</div>	<div>Header (version 1.6)</div>
<div>Revision 1</div>	<div>Revision 1</div>
<div>Revision 2</div>	<div>Revision 2</div>
<div>Body</div>	<div>Body</div>
<div>Stream</div>	<div>Stream</div>
<div>Catalog</div>	<div>Catalog</div>
<div>Page</div>	<div>Page</div>
<div>ObjectStream</div>	<div>ObjectStream</div>
<div>ContentStream</div>	<div>ContentStream</div>
<div>Stream</div>	<div>Stream</div>
<div>Stream</div>	<div>Stream</div>
<div>FormXObject</div>	<div>FormXObject</div>
<div>ObjectStream</div>	<div>ObjectStream</div>
<div>MetadataStream</div>	
<div>ObjectStream</div>	<div>ObjectStream</div>
<div>ObjectStream</div>	<div>ObjectStream</div>
<div>XRefStream</div>	<div>XRefStream</div>
<div>Trailer</div>	<div>Trailer</div>

The successful sanitisation of metadata was also confirmed by checking the PDF document's properties with Adobe Acrobat Reader as shown below. Empty metadata fields are highlighted.

Document Properties

Description | Security | Fonts | Custom | Advanced

Description

File: test7\_cute\_Redacted\_san.pdf

Title:

Author:

Subject:

Keywords:

Created:

Modified:

Application:

Advanced

PDF Producer:

PDF Version: 1.6 (Acrobat 7.x)

Location: C:\Users\User\Desktop\PDF\PDF Project (Final)\Analysis\Test 7\

File Size: 16.13 KB (16,513 Bytes)

Page Size: 8.26 x 11.69 in

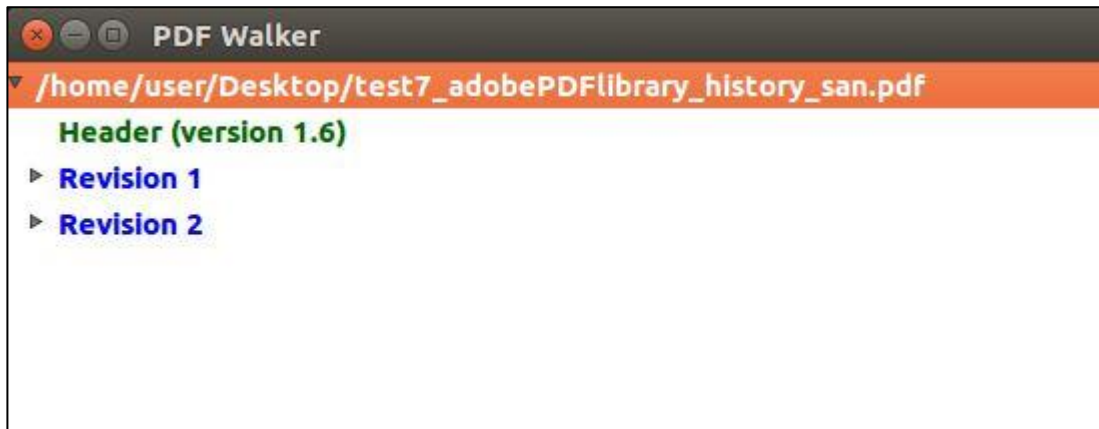
Number of Pages: 1

Tagged PDF: No

Fast Web View: Yes

This test also investigated whether sanitising a PDF document would affect the CMap data remnants identified in previous tests. The PDF documents that were rendered with CutePDF and LibreOffice Writer were sanitised and the resultant PDF documents were parsed with the PDF Stream Dumper tool. In both cases, this had no effect on the CMap remnants in the redacted PDF documents.

Furthermore, this test also investigated whether sanitising a PDF document would remove multiple historical revisions of text as seen previously in Test 3. When the PDF documents were sanitised and parsed with the pdfwalker tool, it was evident that historical revisions of text had been removed. The structure of the sanitised PDF document is shown below.



## Appendix B: Discussion of testing results

It was demonstrated that there was a difference between the CMap objects generated by the different rendering engines. Microsoft Word did not embed a custom CMap and Adobe Acrobat Distiller only did so for a single character.

Adobe Acrobat, CutePDF and LibreOffice Writer all embedded custom CMap objects. Adobe Acrobat did not customise the order of character code to character selector mappings and instead used the order of mappings as they appear in Unicode. In contrast, CutePDF using open source Ghostscript and LibreOffice Writer both customised the order of mappings so that it reflected the order that characters first appeared in text. Thus, the order of mappings in the CMap objects created an encoding mechanism from which meaningful data could be extracted.

Within CMap data structures, mappings are created the first time a character appears in a per-font context. The CMap will remain unless all characters of a particular font are deleted from the PDF document. If a single character remains, the CMap will remain. Thus, if redaction removes all characters that are mapped within a CMap, it can be expected that the CMap will be deleted and redaction will be successful. This remains to be tested. In this case, the analysis only partially redacted the text and this left the CMap objects in place.

PDF documents created with Adobe Acrobat, Adobe Acrobat Distiller or Microsoft Word were able to be successfully redacted by Adobe Acrobat. Parsing of PDF documents failed to identify any remnants of redacted data. This is a result of the fact that these rendering engines either did not use embedded CMap objects or if they did, the order of mappings did not reflect the order that characters first appeared in text. In contrast, PDF documents created with CutePDF or LibreOffice Writer were not successfully redacted. Parsing of these PDF documents found remnants of redacted data within CMap data structures.

Data remnants that were found in redacted PDF documents were the result of:

- the rendering engine creating CMap objects in which the order of mappings was determined by the order that characters first appeared in text
- redaction failing to reorder the mappings within CMap objects
- redaction failing to delete orphaned mappings for characters that no longer existed
- CMap objects remaining as all text of a particular font was not redacted.

This represents a vulnerability which occurs if the following pre-conditions are met:

- the PDF document was rendered by CutePDF (Ghostscript) or LibreOffice Writer
- Adobe Acrobat was used to redact the PDF document.

The PDF standard was checked for a requirement in regard to input characters and the order of mappings within a CMap object but none was found. Neither was a requirement found within the [Adobe CMap and CIDFont Files Specification](#). Adobe has released a [developer technical note](#) that specifies that the order of character codes in a CMap must be in increasing byte order but not how the order of mappings relates to input characters.

If the relationship between input characters and the order of mappings within a CMap is arbitrary, it explains the results in this document and means that custom mechanisms are not prohibited by any specification. It might also mean that the redaction software should be responsible for identifying and removing artefacts of redacted data from a CMap object. In this regard, the [Application Software Extended Package for Redaction Tools](#) protection profile, published by the National Information Assurance Partnership (NIAP), is ambiguous. There is a requirement for the target of evaluation to remove all references and indicators in the structural data to objects that are 'completely redacted'. If text is partially redacted on a per-font basis, this could mean that data remnants in a CMap would be allowed. The protection profile hasn't been assigned to any software products.

Since CMap objects are created on a per-font basis, the likelihood of recovering remnants increases the closer the redacted text is located to the first incidence of a character from a particular font. In addition, it was observed during testing that mappings within CMaps were unique, although there is no requirement that this be the case. As a result, no matter how much previous text exists on a per-font basis, the likelihood of recovering remnants increases if the redacted text is composed of unique characters that occur for the first time. The likelihood of recovering remnants also increases as the amount of text in the PDF document decreases.

It is simple to demonstrate this vulnerability in a test environment. In real-world PDF documents however, the likelihood of recovering redacted text from CMap objects would vary. Real world examples that could be vulnerable to the recovery of remnants might include:

- redacted text occurring at the beginning of a paragraph where a font is used for the first time (e.g. at the beginning of information quoted from another source which is highlighted via use of a different font)
- the redacted text is a password, key or passphrase that is comprised of unusual characters that occur nowhere else within the PDF document
- a small PDF document with very little text.

This vulnerability could be mitigated if Adobe Acrobat's redaction functionality:

- randomised the order of mappings in CMap objects or used a pre-existing order such as that found in Unicode
- parsed CMap objects and deleted orphaned mappings.

This vulnerability could also be mitigated if the CutePDF and LibreOffice Writer rendering engines changed the order of mappings in CMap objects so that it did not reflect the order in which characters first appear in text.

Due to the above, the highest assurance that remnants of redacted data will not remain in PDF documents requires organisations to:

- verify the original PDF document was created using Adobe Acrobat, Adobe Acrobat Distiller or Microsoft Word by checking the metadata of the PDF document
- perform redaction and sanitisation of the document using Adobe Acrobat Pro DC 2017.