Australian Government
Department of Defence
Intelligence and Security

PROTECT

An Examination of the Redaction Functionality in Adobe Acrobat Pro

Defence Signals Directorate - Network Vulnerability Operations

November 2011

Contents

Executive Summary 3
Scope 3
Findings 3
Test Document 4
Test Case 1 – Redaction of Embedded Text 5
Test Case 2 – Redaction of Embedded Graphics 8
Test Case 3 – Redaction of a File Containing Historical Objects 9
Test Case 4 – Redaction of Form Data 10
Test Case 5 – Redaction of Obscured Content 10
Test Case 6 – Redaction of Encrypted PDF Files 11
The “Remove Hidden Information” and “Sanitize” Features of Acrobat 11
Examples of Poor Redaction Techniques 12
Example 1 – Drawing a Box over Sensitive Text or Images 12
Example 2 – Simple Deletion of Text 12
Example 3 – Changing Text/Background Colour to Obscure Content 12
Appendix 1 13
References 13
Tools Used for Analysis 13

NOTE: THIS DOCUMENT CONTAINS PRELIMINARY TECHNICAL ANALYSIS FOR IMMEDIATE RELEASE TO APPLICABLE PARTIES. SUBSEQUENT ANALYSIS OR ADDITIONAL INFORMATION MAY CHANGE THE CONTENT, COMMENTARY AND CONCLUSIONS DETAILED. THIS DOCUMENT DOES NOT CONSTITUTE A DSD CERTIFICATION OR FORMAL EVALUATION.

Executive Summary

DSD performed analysis on the redaction functionality provided in recent iterations of Adobe Acrobat Pro. This functionality allows a user to remove sensitive content from documents prior to dissemination. DSD performed this analysis to gain some assurance over the redaction functions due to prior public incidents where seemingly redacted information was easily recoverable. **Examination of sample redacted documents indicated that when applied as suggested, the redaction function in Acrobat successfully removed information. It was not possible to recover data which had been redacted.**

Scope

This document is not intended to be a complete analysis or evaluation of the portable document format (PDF), its security features, or the security of Adobe's products. It is also not a guide on implementing the redaction function. For the purposes of this document, the following scenario was considered:

- A user possesses a PDF file which they intend to disseminate. However, prior to dissemination, they wish to redact sensitive information contained in the document.
- The user redacts the information using the function contained in Adobe Acrobat, following the procedures suggested by Adobe.
- They save the redacted document as a new file and it is disseminated.

DSD analysed the new, redacted file in order to determine whether the redaction process was successful or if it was possible for a user to view the sensitive information.

Simple PDF files were prepared as test samples using Acrobat Pro 10 and Microsoft Word 2010. These documents contain a title and two lines of text, one of which is to be redacted. A number of test cases were considered, representing the different means by which textual data can be stored within a PDF. These cases were repeated using encrypted PDFs and files containing version histories of past edits. Whilst not explicitly tested, the sanitization features of Acrobat are also briefly discussed. A screenshot of one of the sample files is attached later in this document.

Adobe Acrobat Pro release 10.0.0 was used to conduct the redaction in all test cases.

Adobe's recommendations for using the redaction tool can be found in Appendix 1 - reference 1. A full description of the PDF specification and its history can also be found in references 2 and 3 of Appendix 1.

Findings

Using DSD's test cases, the redaction function worked as intended and it was not possible to retrieve the sensitive information from the redacted document. Rather than simply "covering up" embedded text or field data, Adobe's redaction function removed it entirely. Similarly, when the process was applied to embedded graphics the graphic data itself was altered rather than another graphic, such as a black box, simply being placed on top of it. In cases where the test document contained multiple instances of its data objects due to the use of incremental updates, the redaction function removed

the historical objects and they were not recoverable. When redactions were applied to a document, a new document was created from scratch rather than a save being applied to the original. The "remove hidden information" and "sanitize" functions within Acrobat also successfully removed metadata and historical objects when applied to documents being edited, but not redacted.

Test Case	Successful Redaction
Data stored as embedded text	✓
Data stored as embedded image	✓
PDF contains historical data	✓
Data stored in form field	✓
Data obscured by other objects	✓
PDF is encrypted	✓

Test Document

The screenshot below shows the test document used to conduct the analysis:

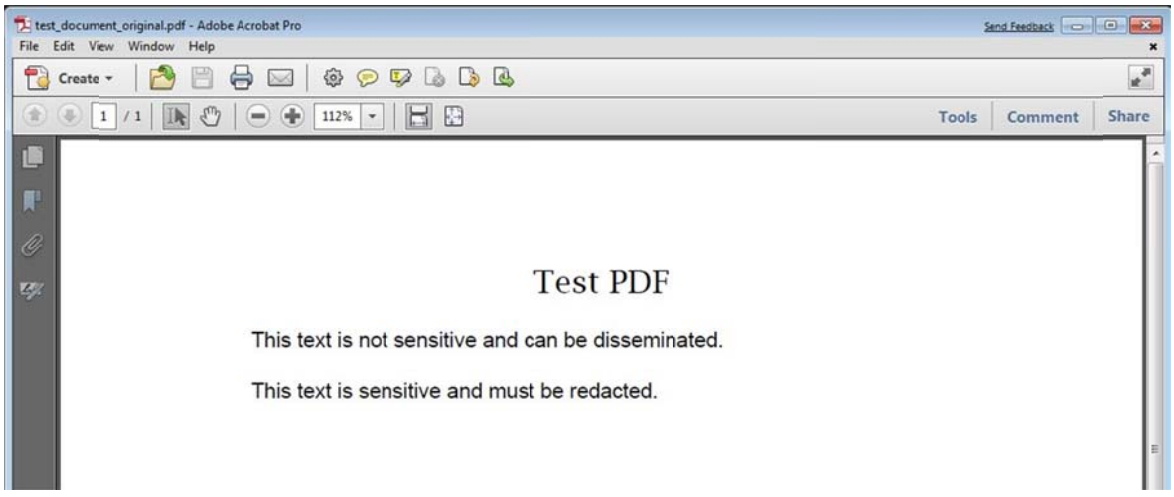

Figure 1 – Example of test document prepared.

Test Case 1 – Redaction of Embedded Text

PDF files store data as a series of referenced “objects”. The files are primarily text- based though possess the ability to store data as compressed binary streams. Prior to redaction, a test document storing the data as selectable text was pulled apart using the open-source tool `pdf-parser`, decoding all embedded streams and viewed in its raw form:

```
./pdf-parser.py –f –-raw test_document_original.pdf
```

Other tools, such as `Origami`, `pdfminer` or `pypdf` would achieve the same result.

Searching through the decoded file, it was determined that the text data was stored within Object 36. The relevant portion of the decoded data stream within this object is shown below:

```
BT
0 g
/TT0 1 Tf
0 Tc 0 Tw 0  Ts 100  Tz 0 Tr 18 0 0 18 267.469 705.691 Tm
[(Test )1(PDF )]TJ
ET
/LEP BMC
EMC
EMC
/Span <</MCID 1 >>BDC
BT
/TT1 1 Tf
12 0 0 12 306 688.925 Tm
( )Tj
ET
/LEP BMC
EMC
EMC
/Span <</MCID 2 >>BDC
BT
/TT1 1 Tf
12 0 0 12 108 674.525 Tm
(This text is not sensitive and can be disseminated. )Tj
ET
/LEP BMC
EMC
EMC
/Span <</MCID 3 >>BDC
BT
/TT1 1 Tf
12 0 0 12 108 660.125 Tm
( )Tj
ET
/LEP BMC
```

```
EMC
EMC
/Span <</MCID 4 >>BDC
BT
/TT1 1 Tf
12 0 0 12 108 645.725 Tm
(This text is sensitive and must be redacted.)Tj
ET
```

The highlighted portions above were the only instances within the file where the text was stored. It should be noted that PDF files do not always store text as continuous plaintext strings (though this is case in the examples to simplify the demonstration). For example, text can be stored as a series of Unicode values represented in hex of octal notation, or assigned custom character mappings to make the document compatible with other languages. Different pieces of text, which may appear near each other when rendered, can be stored in separate objects. The location of data and objects internally within a PDF does not impact how the data is displayed.

Using the function in Acrobat, the line of text “This text is sensitive and must be redacted” was redacted. The new file was decoded and examined. It was determined the internal structure of the file had changed and the embedded text was now stored as a data stream within Object 30. Again, the relevant section of this decoded stream is displayed below:

```
BT
/Span <</MCID 0 >>BDC
0 g
/TT0 1 Tf
0 Tc 0 Tw 0 Ts 100 Tz 0 Tr 18 0 0 18 267.469 705.691 Tm
[(Test )1(PDF )]TJ
EMC
/Span <</MCID 1 >>BDC
/TT1 1 Tf
12 0 0 12 306 688.925 Tm
( )Tj
EMC
/Span <</MCID 2 >>BDC
-16.5 -1.2 Td
(This text is not sensitive and can be disseminated. )Tj
EMC
/Span <</MCID 3 >>BDC
0 -1.2 TD
( )Tj
EMC
ET
q
0 g
1 0 0 1 108 72 cm
/Im0 Do
```

```
Q
q
0 g
1 0 0 1 504 720 cm
/Im0 Do
```

As can be seen above, the redacted text no longer exists within the object. Searching through the entire file, it was determined that legacy versions of the original object potentially containing the redacted text were not present.

The redacted text did not exist within any of object present in the file.

Test Case 2 – Redaction of Embedded Graphics

For this test, data was represented as an embedded graphic where a printed document has been scanned and exported to PDF without any optical character recognition (OCR). The tool `pdfimages` was used on the test file, extracting all embedded graphics in portable pixel map (ppm) format:

```
pdfimages test_document_image.pdf images
```

As expected, this resulted in one graphic file being extracted. Additionally, the file was run through `pdf-parser` to determine where within the file the graphic data was stored. In the test case, the graphic data was stored as an encoded stream within Object 12 as shown below in the output generated by the script. The entire file was subsequently decoded and searched. No other instances of the data were identified.

```
…
Indirect Object: 12
3: 7,14,10
/Catalog 1: 8
/Metadata 2: 11, 1
/ObjStm 2: 2, 3
/Page 1: 9
/XObject 1: 12
/XRef 2: 13, 4
```

The file was opened in Acrobat and in this case, since the text itself was not selectable, a graphical area was highlighted for redaction. Looking at the structure of the resulting file, only one embedded graphic object was identified. This object was extracted using `pdfimages` and is shown below:

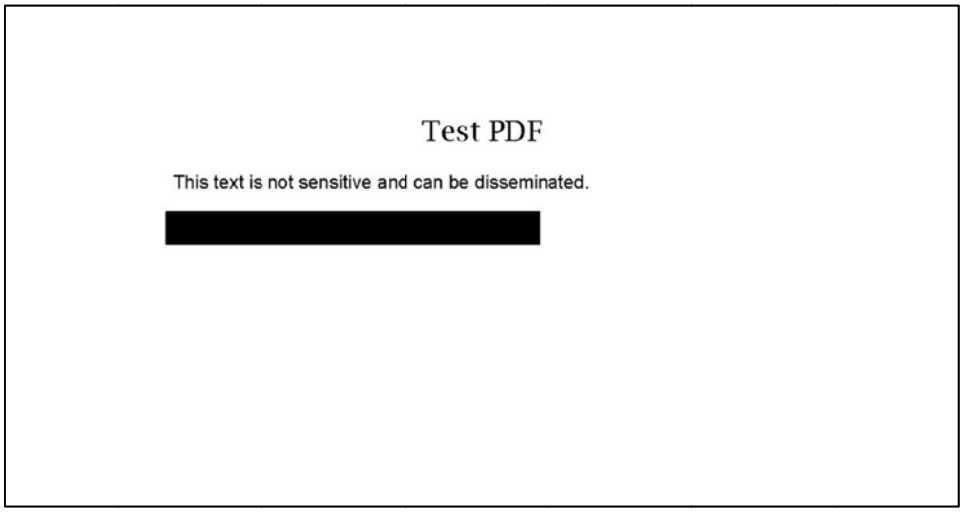

Figure 2 – Image extracted from the redacted file

This demonstrates the raster data of the graphic itself had been altered to include the redaction. A black annotation was not simply covering the original graphic object.

As in the first test case, the internal structure of the new file differed from the original and in this instance, the graphic data was stored as Object 14. An exhaustive search through the file indicated no historical instances of the original, unaltered object.

Test Case 3 – Redaction of a File Containing Historical Objects

The text in the original file was edited and re-saved several times in order to create a document which possessed a number of historical objects containing the previous iterations of the data. The figure below shows this document's structure as displayed by `pdfwalker`, a part of the `Origami` suite of PDF tools.

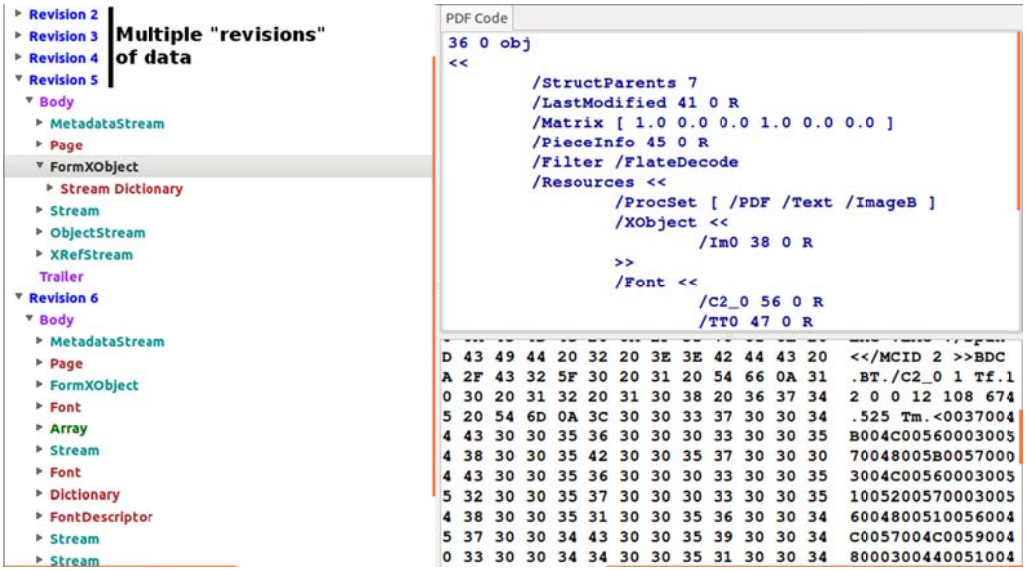

Figure 3 – List of objects within a file containing historical objects.

When redactions were applied to this document and saved, the new file did not contain any of the historical objects present in the original; even if the "remove hidden information" option was not run after the redaction was applied.

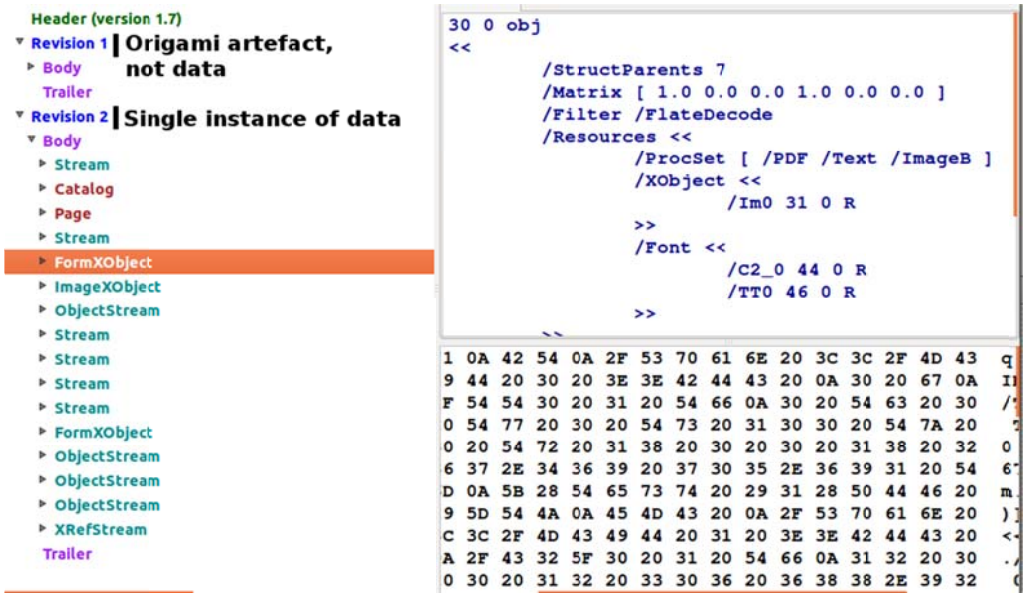

Figure 4 – List of objects within redacted file.

Test Case 4 – Redaction of Form Data

A test document was prepared where the text data was stored as content within a field, simulating an interactive form.

When the internal structure of this document was examined, the field content was found to be in two separate objects, one being a Form object and the other a being an object used for such things as controlling a form's appearance, referred to as a "widget" object.

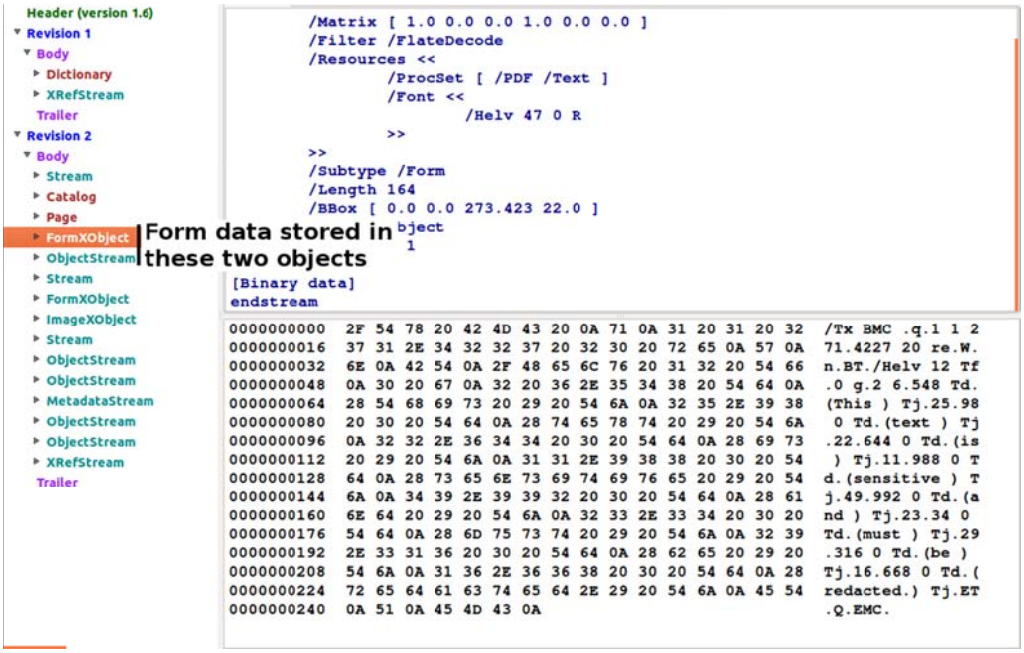

Figure 5 – List of objects when data stored within form field.

Redaction was applied to the entire field content and the new file examined. Both instances of the data had been removed and no remnants of the form data were discovered.

A second test was conducted with redaction applied to only part of the field content. In this circumstance the entire field was still removed by Acrobat during the redaction process, including content not marked for redaction.

Test Case 5 – Redaction of Obscured Content

Test case 1 was repeated; however a drawing was first placed over the text. This was done to confirm that when an area of a page is marked for redaction, all content, not just the top layer is removed.

The area containing the appropriate text was marked for redaction. The resulting file was examined for the redacted text. The text data, as well as the drawing covering it, were not present in the file, indicating that redaction can remove multiple layers of data.

Test Case 6 – Redaction of Encrypted PDF Files

Test cases were repeated using encrypted PDF files requiring a password to open. In these instances the redaction results were the same as above. Additionally, the encryption was maintained in the new file after the redaction function was run.

The "Remove Hidden Information" and "Sanitize" Features of Acrobat

These two functions scan a document for any objects that are not rendered when it is displayed. They identify numerous items including, but not limited to: metadata, objects that are transparent or completely covered by other objects, historical objects still present after edits, any un-referenced data in the file, JavaScript, hyperlinks and file attachments. When applied, these functions remove these items from the file.

A complete test of these functions is beyond the scope of this assessment; however, in the instances when the functions were applied to test documents, they were successful in removing metadata contained within the files.

It is good practice for users to enact these features when conducting redactions in order to ensure unnecessary information is not still present. When redactions are applied, Acrobat should be set to prompt the user to run the "remove hidden information" function.

It is also recommended users run these features after conducting edits on a PDF to ensure that information contained in previous iterations of the file is thoroughly removed and not accessible (see example 2 of poor redaction techniques below).

Examples of Poor Redaction Techniques

The examples below show common, easy to circumvent, ways users have chosen to redact information. Whilst it may seem obvious why these techniques do not work, there have been some very public incidents involving improperly redacted documents (Appendix 1 - references 4 and 5).

Example 1 – Drawing a Box over Sensitive Text or Images

It is quite common for users to redact information in a PDF file by simply drawing a coloured box over the sensitive data using the PDF editor available to them. Whilst the sensitive data is obscured when the document is displayed, it is still present within the internal structure of the file.

Using `pdfimages`, it is a trivial matter of extracting images from within a PDF file. Since the addition of a coloured box does not alter the original embedded image, when `pdfimages` is run on the "redacted" document, the original, pre-redacted graphic is retrieved.

In the similar case of a box drawn over embedded text, the obscured text is easily recovered by extracting the contents of the objects stored within the file using one of the many tools available, or by simply "selecting all" and pasting the text somewhere else.

Example 2 – Simple Deletion of Text

It would make sense to a user that deleting a line of text from a document would be an acceptable method of removing the data. However, the PDF specification allows historical objects to be stored within the file after they have been edited. If an object is altered, a new version of that object is created. The original object is still present in the file, but the new object is listed in the file's reference table and thus rendered when a user opens the document. This is a process known as "incremental updating". The original object can be recovered by examining the internals of the file.

This problem can be circumvented by running the "remove hidden features" function over the document after editing is complete.

Example 3 – Changing Text/Background Colour to Obscure Content

There have been instances where users have redacted information by changing the text background to match the text colour. This is the simple to circumvent; a user can simply highlight the obscured text with their cursor to make it visible. The examples listed in references 4 and 5 both used this "redaction method".

It should be reiterated that regardless of the software used, for a redaction method to be successful it needs to remove the data, not simply obscure it.

Appendix 1

References

1. Acrobat X Pro / Remove and Redact Information: http://www.adobe.com/products/acrobatpro/pdf-redaction.html
2. The PDF Specification and Reference 1.7: http://www.adobe.com/devnet/pdf/
3. History of the PDF: http://www.adobe.com/pdf/about/history/
4. How NOT to Redact a PDF – Nuclear Submarine Secrets Spilled: http://nakedsecurity.sophos.com/2011/04/18/how-not-to-redact-a-pdf-nuclear-submarine-secrets-spilled/
5. AT&T Leaks Sensitive Info in NSA Suit: http://news.cnet.com/2100-1028_3-6077353.html

Tools Used for Analysis

Tools used to conduct the analysis are listed below. They are all freely available online.

- pdfminer – 20110515 – http://pypi.python.org/pypi/pdfminer
  Python scripts for extracting information from PDFs.

- pdf-parser – 0.3.7 – http://didierstevens.com
  Python script for examining the internal structure and content of PDFs.

- Origami – 1.0.4 – http://www.security-labs.org
  Ruby based set of tools for extracting data from PDFs.

- pdftk – 1.44 – http://www.pdftk.com
  A package of small tools for extracting information from PDFs.

- pyPDF – 1.12 – http://pybrary.net/pyPDF
  Python library with functions for creating and extracting data from PDFs.

- pdfimages – 3.00 – http://poppler.freedesktop.org
  Image extraction tool derived from xPDF.