Faith in the Format: Unintentional Data Hiding in PDFs

Matthew Ryan Davis
mattdavis9@gmail.com
A 757Labs.com Effort

August 3, 2008

Abstract

Adobe's Portable Document Format (PDF) has gained a prominent foothold as a method of distributing text-based and graphic-based information. Its use has become ubiquitous across academic, technical, and governmental institutions and has become one of the forerunners of information dissemination. The success of this format can be thanked, in part, to the designers and their creation of a platform-independent method for rendering text and graphics. Such independence allows users to feel assured that their data is viewed similarly to the original document, despite the platform the reader is consuming the information upon. However, that trust is derived from the faith in the format through the tool used to read the data, even if there is more attributes hiding under the covers. When a document is created, modified, and re-saved, Adobe's standard allows for previous versions to be retained, in essence creating a running history of the document. Such revisions, while not seen in the distributed copy, still reside in that document. The basis of this paper is focused on the concept that document maintainers might not be aware of this property, and consequently distribute modified versions containing non-public information. With such a high usage of this format, it would not be a surprise that there are numerous public documents containing non-public data that is merely non-visible but still accessible.

1 Introduction

The success of a file format can be attributed to the frequency of use. Such frequency is typically a function of convenient features that make-up the format. However, some of these features are not always apparent to the end-user, and some might argue that a bit of sheltered ignorance is not a bad thing. APIs use techniques of information hiding, encapsulation, to prevent other coders from using objects improperly and/or compromising the overall integrity of the system in development. Likewise, strategies used behind the covers of a file format do not all have to be understood to be effective. For instance, understanding the Huffman-Coding or frame structure of an MP3 file seems rather irreverent for a casual listener. What does matter is the effectiveness of the format. Does the format do what is expected by the end-user? It is at this point one might question their trust in the format.

In the case of PDF files (Portable Document Format), it seems a rational judgement to assume that a document encoded to PDF specifications will render nearly identical on all intended recipient's machines. After all, 'P' in this case stands for "Portable." The rendering of a PDF is truly the responsibility of both the application that encoded the document and the application reading the document. That trust in creating versus reading is why there is a standard surrounding PDFs.

It is this trust, or faith, of the Adobe PDF format that concerns this paper. Section Two of this document discusses some of the features that has made the PDF format a success, including revision history. Section Three is a brief study regarding PDFs in the public domain. Section Four is a look at other document formats and Section Five concludes this paper.

2 Portability Features Underlying PDF

It might be arguably so that the success of the PDF format is solely based on its portability. No matter what platform a user might be running, chances are that there is a PDF document reading utility available. So what makes this portability a reality? This section discusses some of the features that enable the PDF format to become both an efficient and portable way to distribute information.

2.1 Embedded Fonts

The innumerable amounts of fonts available might first make one queasy. After all, how is the reader supposed to view a document when they can only, at best, have a subset of all available fonts the world over? The PDF format addresses this concern by allowing for the original document's font to be embedded into the resulting PDF document, being that the font is one of the numerous supported formats (e.g. TrueType, Type 1, OpenType). This portable-friendly feature allows a PDF creator to use nearly any font of their choice, no matter how obscure, and removes worry regarding how the resulting document will look on the recipients' machines. This feature requires no intervention by either the developer or reader while also removing any kind of dependence on the machine that is responsible for reading the document. Such functionality does not come without drawbacks, mainly size. Adding fonts to a PDF increases the overall size of the document. However, Adobe allows for data compression of various objects inside a document as a "stream." Therefore, space can be reduced by having the font embedded in a compressed fashion. Consider a document composed of numerous fonts, and the amount of processing that a typical PDF "save" might require. Assume that each fresh save of the document will have to bundle in these fonts so that the resulting PDF is portable. The aforementioned concern for efficiency, during file-saves, plays an important role for the "revision history" discussed later [1].

2.2 Images and Other Media

The ability to incorporate graphics and images alongside text, is a testament to the universality of this format. By graphics, I elude to the primitive constructs the standard defines, which allows for the drawing of lines and text. Images, on the other hand, can be "content streams" similar to font data. It is important to note that by viewing the images of a PDF document, one

places a certain amount of faith that the PDF viewing application renders the data to the likes of the original.

The standard also increases the portability of the format by allowing other types of media to be embedded inside the document. Media along the likes of movies, slideshows, interactive forms, and even 3D-objects are allowed. Details of such are in the standard, however it should be noted that all media is the responsibility of the viewing application, whether that viewer leverages other libraries or tools to render the document is at the rendering application's discretion. In effect, there is a lineage of trust that exists between the viewing application and utilities harnessed in rendering any part of the PDF. This lineage further abstracts the end-user from what is really going on, and even increases the potential for security exploits. Sharing processing duties amongst other libraries and tools is nothing unique to PDF, and has become ubiquitous in the information age, as such compatibility and reuse aids in arriving at the end result in the most efficient and convenient manner. After all, there is no need to re-invent the wheel.

2.3 Compressed Data

As one can imagine, a document containing multiple fonts, images, pages, and media can grow quite unwieldy. To ameliorate what might otherwise be a large document in size, objects composing the PDF can be compressed. Even a series of objects, also known as an "object stream," can be compressed. With the ability to compress data, document size was probably one of the original concerns when the format was originally considered.

2.4 Revision History

While maintaining a document that might be rather large in size, one might also question the efficiency of saving data. Especially if certain portions of the document might undergo an encoding process. When a document undergoes a modification, chances are it is not the entire PDF that needs reworking. For instance, maybe a short and simple text edit to correct a misspelling was all that was needed. In such a case, and especially for large documents, re-encoding the entire document, images and all, could produce a lot of redundancy and wasted cycles. To reduce such redundancy, the PDF specifications allow for the document to be appended to. In other words, only the objects that undergo a change need to be re-created. The object's original and unmodified brethren still exist, however the PDF viewer need not reference that older object's versions.

Appending changes to the document, instead of recoding the whole work all over again, produces a running history of what was changed. This information is then passed on when the document is distributed. These changes are not limited to modified objects, but also incorporate removed objects. It is my personal feeling that this is a rather "unknown" attribute of the PDF format, and is easily justified by the logic that it creates efficient document re-saves and provides a history of changes for undoing and comparing.

The PDF 1.7 specification states:

> "In addition, because the original contents of the document are still present in the file, it is possible to undo saved changes by deleting one or more addenda. The ability to recover the exact contents of an original document is critical when digital signatures have

been applied and subsequently need to be verified."

Appending modified data, instead of overwriting, eases document maintenance. Each object in the document is referenced by an offset of where it appears in the file. All of the object offsets are easily located in a cross-reference table that is stored at the very end of the document. As the document becomes modified, only an updated cross-reference table needs to trail the document. Consider what would happen if an object were to be modified in place, instead of appended. The result would be that the offsets of all subsequent objects would have to be recalculated and the cross-reference table updated. Likewise, more space might be needed, and would result in a new file with appropriate memory allocated to rewrite the changes and all subsequent objects. Appending, versus in-line object updating, also avoids misalignment of the cross-reference table, as no object offsets have to be recalculated. Therefore, appending objects is the easier and more efficient approach to document updating.

There are cases when the unmodified information is not desired. Consider removing information from a document that might have been classified knowledge, and then making the document available in the public domain. It is possible to save a modified document without producing the extraneous information that might have sensitive data. However, that role lies at the mercy of the PDF encoding application and its user.

3 PDFs in the Wild

Being that PDFs retain modified and removed data from previous versions, I decided to conduct a very brief study of PDFs that are available in the public domain. I was not terribly sure what I would find, however, I was hoping to find documents that contained multiple revisions.

3.1 Tool: pdfresurrect

To aid my efforts in examining the PDFs available in the public domain, I decided to write a small application[1] that produces all versions of the PDF in question. To accomplish this task, the tool merely looks for the PDF end of file tag (%%EOF) that gets created after each document revision. The tool then gathers all of the cross-referenced objects, which are later used to produce a summary of the analyzed document. Instead of building a fresh PDF for each version, the tool takes advantage of the file format and appends to the end of the file the cross-reference table that was created for that version. That table, as mentioned earlier, contains the object offsets for the version in question.

Once all versions have been written, a summary file is produced. This summary uses the cross-reference tables for each version, starting with the least recent, and proceeding to the final version, in order to generate a list of all objects and provide a object-modification status: (A)dded, (M)odified, or (D)eleted. The object type, if found, is also reported. Since a PDF consists of multiple object types, changes to documents might only be aesthetic (e.g. font, pagination) and not the text changes that some might find more informational.

3.2 Study

The study was conducted by obtaining 50 PDF-based documents per 5 top level domains chosen

[1]This tool can be downloaded from:http://www.757labs.com

(.com, .edu, .org, .gov, .mil). This data gathering resulted in a total of 250 PDF documents collected for the purpose of examining revision history. To remove any personal bias, the documents were obtained as the first 50 results pulled from a Google query for the said filetype and domain. In the case of a dead link, the next document from the result set was pulled.

After the sample data was obtained, the pdfresurrect tool was then used to obtain history information from all of the documents. For the purposes of this study, the main goal was to obtain a ratio of non-modified to modified documents (i.e. those with stored previous revisions). Such data should give a rough idea as to how much history is being distributed, possibly unknowingly, in the public domain.

3.3 Results and Mitigation

The following list shows the findings resulting from the obtained documents:

.com: 0 of 50

.edu: 2 of 50

.org: 2 of 50

.gov: 0 of 50

.mil: 2 of 50

The results produced ratios that were less than expected, and are merely showing that the more popular of results from Google have little retained history information. These findings do not elude to personal documents distributed amongst cohorts, or businesses disseminating documents amongst their employees. Of concern might be classified documents that have varying levels of classification, that were not picked up on the rather small data gathering. Such a case would be if a document was originally classified as "top secret" and then being modified to produce a "secret" version. Obviously, the institution scrubbing the "top secret" data would not intend to have that information distributed to those authorized at the lesser "secret" level.

Some of the information from obtainable histories did result in a few unique findings. Nonethe-less, this brief exploration has proven that multiple revisions do exist in the public domain, and it is expected that a more thorough investigation should generate greater findings, especially when the search has a targeted focus, such as business intranets.

Fixing such a potential problem is rather easy. One simple mitigation that can prevent the distribution of past revisions, is to merely cut and paste the data as a new document and then save that version. The resulting document would be generated as if it were created from scratch, and never have the chance to acquire history information. Another mitigation would be to use the aforementioned pdfresurrect tool, which has an option to write a character over data for the original document objects that have been modified.

4 Other File Formats

While this paper is primarily concerned with information hiding resulting from the maintaining of revision history in a PDF, it only seems appropriate to explore other formats of the like. Two common formats, the open OpenDocument and Microsoft's Word, are used in similar fashion as the PDF and will be briefly explored in this section. These formats do retain document revision information. This paper is not concerned with

the utility that can be harnessed from retaining revision data. Of concern is the idea that not all document maintainers are aware that such information is being saved, and thus might reveal information that should otherwise be squelched.

4.1 OpenDocument Format (ODF)

The OpenDocument format is an open standard format maintained by the OASIS group. This format encompasses a wide gamut of document types common, but not necessarily limited to, office applications (e.g. word processing, presentations, charts, spreadsheets, drawings) [2]. This format is incredibly flexible, in part to the fact that it is defined as an XML schema. Not only does the format define the various elements viewed when creating the document (e.g. text, paragraphs, images) but there is quite the amount of metadata that can underlie the document. Such information contains creation dates, user modifications, etc. To further flexibility, the metadata can be custom or user created [3].

Upon initial glance at this standard, the document has the ability to manage quite a bit of metadata about itself. It only goes to follow that the method of maintaining revision history is well-defined and incredibly helpful when trying to determine what changes were made. First, the ability to "track" document changes can be toggled on or off. Unlike PDF, the authoring tool can enable or disable this history tracking feature. The functionality of the PDF is based heavily on file offsets of the objects. If a PDF object were to be removed completely, the cross-reference table would not match the objects in the document, and consequently become misaligned. Another rather helpful feature of the ODF's history tracking is that it tells what text data was changed, by whom, and when. It is important to note here that the document changes that have been mentioned regard text documents, and not graphics. The specification states that changes of table data in spreadsheets are tracked but not in common text documents. None-the-less, the potential for encapsulating sensitive information from previous revisions can be retained in such documents.

4.2 Microsoft Word '97-07 Binary (.doc)

Microsoft's Office, a suite of office applications, has been an incredibly popular toolset for nearly any domain (home, academia, work, etc). Many institutions require the use of such an application set, even though there are comparable alternatives available that can read and write this format. The format of brief discussion here covers the Word '97 Binary (.doc) format.

As expected, features of Word are aided by retaining metadata and past revision information within the document innards. Such features as document merging, annotations, and revision tracking, seek benefit from this retained information. Files of this format contain a flag that can be set allowing recipients of the document to have their changes untracked. Unlike a PDF document, Word's binary format has its main information header at the beginning of the document. The Word Binary specification defines two modes of saving a document. The first being a traditional "full-save" and the other being a "fast-save." The fast-save, according to the specification, appears to retain previous document information and merely appends the updated or added information to the end of the document. This latter method is faster than the former as the entire document does not have to be re-saved, merely appended to [4].

5 Conclusion

As one can see, it is easy to place faith in a format and assume it does just what it looks like it should be doing. As we have seen, what you see is not always what you get, and the study shows that a small search of the public domain can turn-over documents that have hidden information. However, quantifying the sensitivity of someone else's revision information is beyond the scope of this study, and making assumptions at that level would only tarnish this paper. What is of concern is not the useful feature of document revision history tracking that Adobe, Microsoft and the OASIS OpenDocument Format leverage, rather the concern is directed towards the document maintainer who might have unintentionally retained sensitive material in a publicly published document.

References

[1] Adobe Systems Incorporated. *PDF Reference Sixth Edition: Adobe Portable Document Format*. Adobe Systems Incorporated. Version 1.7. November 2006.

[2] opendocument.xml.org *ODF Wiki Knowledgebase: OpenDocument Overview*. OASIS. http://opendocument.xml.org/overview February 2008.

[3] OASIS. *OpenDocument Format for Office Applications (OpenDocument) v1.1 Oasis Standard*. OASIS. February 2007.

[4] Microsoft. *Microsoft Office Word 97-2007 Binary File Format (.doc) Specification*. Microsoft Corporation. 2007.