Scalable Exploitation of, and Responses to Information Leakage Through Hidden Data in Published Documents

Simon Byers
byers@research.att.com

2003/04/03

Abstract

In considering the leakage of information through hidden text mechanisms in commonly used information interchange formats we demonstrate how to automate and scale the search for hidden data in Word documents. The combination of this scaling with typical behaviour patterns of Word users and the default settings of the Word program leads to an uncomfortable state of affairs for Word users concerned about information security. We discuss some countermeasures employable by users and note more general consequences of these effects.

1 Introduction

It is well known amongst the more technical computer user communities that documents written and stored in the Microsoft Word™ document format may contain hidden data. Certain segments of non-technical Word users also are aware of this fact, for example many lawyers. We refer the reader to the Risks Archive at `catless.ncl.ac.uk/Risks` for a review of this area. However we feel that awareness of this problem and its generalization is not sufficiently widespread, especially amongst people comprising the bulk of non-technical computer users. Specifically we refer to people who are computer users at work and at home but are not computer professionals and probably not aware of the dangers of opaque proprietary file formats that may carry hidden payloads.

In this paper we demonstrate by implementation the feasibility of scaling the general exploitation of this hidden text effect. The tools we use are simple and do not require specialized tuning or highly skilled input. Throughout we have and assume no knowledge of the MSWord™ document format itself, reinforcing the simplicity of the techniques. Our implementation is specific to MSWord™ but the techniques could be extended to deal with other file formats where such opportunity for hidden data exists, and hence we wish to warn of the more general dangers. The techniques we use are also employed frequently in *Run-the-Business* automated data processing efforts and as such have considerable use in many areas.

After discussing various types of hidden text and hiding mechanisms we discuss the tools required to uncover it and the implementation of our methods. After discussion of our results and higher level applications of this technique we discuss responses to the problem and draw some conclusions.

2 Hidden Text

We expect to find hidden text of the following types based upon anecdotal reports:

- Names and usernames of the document creator and their collaborators.
- Organizational information of the users involved.

- Text specific to the MSWord™ program version and document format.
- Pathnames of the document in the filesystem of the operating system upon which the document was composed.
- Information about the hardware upon which document was composed
- Printer names and information.
- Email headers or web server information.
- Text that has been deleted from the document at some point prior to the save.
- Text potentially from completely unrelated documents that is only present due to a bug in MSWord™ , as opposed to proclivities of the format itself.

We clarify the last type by referring to an incident made aware to us in [1]. In this case the protagonist had two separate and unrelated MSWord™ documents open. He then saved and mailed one of them to a mailing list. One of the members of the mailing list noticed (through use of a non-Microsoft document viewer) that the second document was actually hidden inside the file as well as the intended visible one. In terms of data leakage this type of occurrence can be more catastrophic than most conventional hidden data mechanisms but we do not know whether it is common.

3 Tools

For the benefit of the section of the audience with most to gain from this paper we first describe the basic tools that will be used in our procedures.

- antiword, available from `www.winfield.demon.nl/`, is a free open source command-line tool that turns a MSWord™ document into plain ascii text. It has various options, one of which will be employed directly.
- catdoc, available from `www.45.free.net/~vitus/ice/catdoc/`, is similar to antiword in most respects, but sometimes gives slightly different results. Both of these tools are routinely used by Unix users to view MSWord™ documents sent to them by their point-and-click oriented colleagues.
- "strings" is a standard tool that parses the data in any file and extracts those segments of the data that are printable strings of normal text characters.
- perl is a script programming language useful for manipulating data.

4 Implementation

Our method consists of three basic steps that we separate both operationally and for the purposes of description here.

4.1 Finding Content

This first task is to automatically find content in MSWord™ document format to examine. Our source is predominantly the open Internet. In order to find documents on the Internet one may construct a custom crawler or use the results of existing crawlers. We choose the latter approach and simply use a mechanized interface to any of a number of available search engines. The result of this search is simply a list of URLs that each give the address of a MSWord™ document on the open Internet.

We routinely use keyword salting [2] in searches of this nature. We note that this search can be implemented with only salt and no other specific constraints except the filetype. Any list of arbitrary keyword salt will suffice in this situation but specific keywords can be used in order to target certain sectors of MSWord™ document users. Example sectors of interest might include governmental, healthcare, educational and corporate and each of these can be targeted effectively with appropriate keywords.

4.2 Data acquisition

For each of the URLs found by the previous step we simply download that document to disk. We store it and all it's known originations indexed by the document's MD5 hash value in a regular filesystem tree. We then invert this mapping for use in searches of our content based on origination. This technique is a routine part of many scaled open-internet data gathering exercises.

4.3 Hidden Data Extraction

Once a sizable collection of documents has been collected various methods for finding hidden text can be applied in an automated fashion. We use a variety of very simple heuristics, most of which are based on a comparison with the ascii text generated from parsing the document with antiword or catdoc. Neither utility are completely reliable in parsing MSWord™ documents so we accept some noise in our findings. This of course is to be expected from an analysis that does not use inside knowledge of the hidden data storage and is of little consequence as any specific example that seems to warrant further scrutiny will be verified more carefully by hand.

Where tables are present catdoc is generally easier to work with, but catdoc seems to fail slightly more often than antiword in general parsing. Whichever is used we denote this output the *reference parse*.

- **Method 1:** We compare the reference parse with a parse taken with the -s option in antiword. This option explicitly tries to display hidden data.
- **Method 2:** For each string found in an application of the strings command to the word document, look for a version of that string in the reference parse. We first regularize both the string and the reference parse for white space and line breaks. Any string that is not found in the reference parse is deemed interesting.
- **Method 3:** For each word in each string in the strings output, tabulate the number of occurrences of that word in the strings output and in the reference parse. Any word which has differing counts in the two is deemed potentially interesting, any word which has zero occurrences in the reference parse even more so.

Some simple post processing can be applied to the results of these heuristics or combinations of them.

4.4 Heuristic Performance

We first examine the type of output the three methods discussed above yield. Specifically:

- **Method 1:** This technique only rarely finds hidden text and failed on canonical examples of our own. The changes it finds can be very small, such as a punctuation change, or as large as several deleted lines. Usernames, pathnames and other meta-data are not revealed by this technique.
- **Method 2:** This method yields several kinds of results. Some are short junk strings and can be discounted. Much username, pathname, printer and email data is exposed in this way in such a manner as to make it instantly recognizable. Lastly this method returns deleted words or lines containing a word that was deleted or altered. On word documents without tables or too much exotic structure both the catdoc and antiword reference parses return the same results without much effort.
- **Method 3:** This method yields some information about the internal structure of the word document as well as deleted text. Such structure can consist of two similar copies of the document text that may differ in some small way. The deleted text shows up most obviously as words that do not appear in the reference parse but that do appear in the strings parse or where the word counts indicate a difference in occurrences is present.

4.5 More Sophisticated Extraction

We have purposefully restricted our methods to very simple analysis techniques for two reasons. Firstly we wish to demonstrate how easy this is to implement and secondly to de-mystify this work so that regular MSWord™ users can understand the problem and personally *assess their risk*. We do note however that custom MSWord™ document disassemblers could easily be written starting from scratch or from the source code of existing utilities such as catdoc, antiword or OpenOffice. This requires some knowledge of the document format or some reverse engineering expertise, but many individuals possess these skills. We also note the possibility of connecting to a Windows server and having MSWord™ automatically convert the documents to an open format, but this is generally not robust or fast enough for use on the scales that we find valuable.

5 Results

5.1 General Observations

We obtain MSWord™ documents at a rate of about 1000 per hour through a regular cable modem using no parallelization. Substantial amounts of elapsed time is spent during connection timeout events, so the User Agent timeout should be set quite short and only one or two retries attempted. The first 100000 documents occupy 16G of space.

Some of the retrievals resulted in a file that was either corrupted or not apparently a MSWord™ document, these we simply ignore. All valid MSWord™ documents retrieved have some hidden text in them as found by Methods 2 and 3 although some portion only have the expected MSWord™ related strings and simple meta-data. As mentioned, the application of Method 1 only rarely produced results, in fact in only 75 of the first 100000 documents.

Overall, using a count of words found by Method 3, about half of the documents examined contained between 10 and 50 hidden words, a third between 50 and 500 words around 10 percent having more than 500.

5.2 Examples

We present below an innocuous but real example as an illustration of our findings, which happens to be the first document we processed during initial code testing with Method 2. All names and organizations have been changed to protect the innocent:

```
1|/m/A
3|/o=Fake Unversity/ou=ALUMNI/cn=Recipients/cn=Bill.Fishman
3|/o=Fake Unversity/ou=ALUMNI/cn=Recipients/cn=Angela.Torro
1|3x.2
1|5ZH=Ss
1|8n2NLb
1|A.nH
1|Adobe
1|AiPx
1|AknQ
3|Bill Fishman
1|Bill.Fishman
1|Dh6N
1|Ducky
1|HsoT
1|JFIF
2|John Garnfield
3|Angela Torro
1|Angela.Torro
```

```
1|MSWordDoc
1|Master Privacy Policy for review - hardcopy provided to Bill with signoff sheet
1|Microsoft Word 9.0
1|Microsoft Word Document
1|Normal
1|QDUo
1|Title
1|VFR9
1|Word.Document.8
1|bjbjU
1|dGAMb
1|eNcj
1|koz.com
1|l ja
1|lbXB
1|oVfu
1|tSYd
1|yHYL
```

We include the (sanitized) raw results before post-processing to show the usable nature of the returned results. The names of the three people involved and their affiliations are obviously interesting, as is the deleted notification that this document was once a review copy. Note that Method 1 produced nothing when applied to this example and Method 3 reproduces the above results in a slightly different manner. This particular example has a relatively large amount of the unimportant small junk strings.

The shortest results from the application of Method 2 have about eight returned strings. We show below such an example, again with the name changed:

```
2|John Buskind
1|MSWordDoc
1|Microsoft Word 8.0
1|Microsoft Word Document
1|Normal.dot
1|Title
1|Word.Document.8
1|bjbj
```

There is obviously no interesting hidden text in this document except the name of the composer. Method 3 produces substantially the same results on this example with a couple of extra junk strings.

6 Use in Higher Level Analyses

These techniques can be used as building blocks for higher level analyses of data that may have important end goals with financial and personal consequences. We give two examples here, one benign and one less so to illustrate this.

6.1 Identity Theft

It is simple to examine those MSWord™ documents in our corpus that appear to be resumes and check for a deleted Social Security Number (SSN) hidden in the file. It is feasible that an individual may include the SSN on copies of the resume sent to prospective employers but delete it from the version put online to guard against identify theft. Further it is possible to use a suitable keyword set in the initial document search to target resumes in particular.

6.2 Plagiarism Detection

Using these methods it is possible to gather information on whether a document has been cloned, or plagiarized from another. The computational load can be made manageable by first restricting the search to keywords from the visible text with standard document retrieval techniques but then using the hidden text to gather stronger information.

7 Response to the Problem

7.1 A Sample Event Cascade

In considering responses to this effect we first consider a realistic scenario involving word documents.

- Alice sends a mission critical memorandum written in MSWord™ to Bob, who may be in a different organization or corporation, for example a client of Alice's organization.
- Bob's supervisor, Cynthia, requests that Bob draft the weekly TPS reports.
- Bob, liking the format of Alice's memorandum, simply copies it, deletes text and types the TPS content into the copy.
- Bob sends the TPS report to Cynthia.
- Cynthia places the TPS report on the external web.

This event cascade shows how simple everyday actions can inadvertently place sensitive data squarely in the public domain. Many MSWord™ users behave as Bob does in our example, and even if they did not there is the interesting effect described in Section 2 to contend with. Coupled with our demonstration of scalable access to these hidden data once on the open internet this completes the leakage path of the data from initial user to end abuser.

7.2 Recommendations

Our first and simplest recommendation is to cease all use of MSWord™ for any data that one might not want eventually made public. Our experience with normal corporate information technology standards and user training mandates us to consider softer measures.

Such a measure might be to impose use of a MSWord™ document scrubber by all personnel within an organization. Such a scrubber can be implemented as a desktop tool or as a firewall filter. Such desktop utilities exist and their use is currently recommended in certain circles, for example by some lawyers. A primary problem for the desktop tool is enforcing adoption of this procedure, as with all policy mandated security procedures, if it involves an extra step it may not get used unless technical measures assist with compliance.

Another problem with this approach is that even if Alice (in the example above) scrubs her outgoing MSWord™ content, she cannot be sure her content will not be leaked unless she trusts Bob to also scrub his content. Bob, recall, could be part of another organization entirely and hence not subject to Alice's corporate security rules. We also note that if Alice sends Bob plain text there is still the chance he will convert that into a MSWord™ document and end up releasing the data inadvertently.

Some organizations already have specific recommendations on the use of MSWord™ in order to expose and then remove hidden text during general practice using nothing but various options and actions in MSWord™ itself, e.g `www.mltweb.com/prof/testdoc.doc`. Some of the text purposefully hidden in that document as an example is trivially found using our tools, the output of Method 2 is shown below:

```
1|A sample document which explains why people should send electronic
    files to vendors without looking at it first.
1|Client
```

```
1|Date completed
1|Department
1|Destination
1|Disposition
1|Hanford
1|MSWordDoc
1|Microsoft Word 9.0
1|Microsoft Word Document
1|PC information
1|Something for everyone to read
1|Taylor
1|Title
1|Word.Document.8
1|bjbj
2|h0044311
1|normal.dot
1|somewhere over the rainbow
1|trash
```

The text not found in this case by our methods is that purposefully hidden by the author, and hence potentially of less interest due to the fact that the author is aware of its existence.

The specification of a corporate security policy to address this problem is then somewhat difficult. As part of our everyday work this problem has shown itself to be important as our employer, AT&T, has thousands of MSWord™ documents, large amounts of proprietary data, large numbers of industry collaborators and extensive material on public websites. The collaboration of several sub-organizations needs to be secured for any useful impact to be made in preventing leakage through this channel.

8 Conclusions

In this paper we have shown how to scale the exploitation of a known problem and have noted some patterns of use related to the problem. The simplicity of this scaled search method is cause for concern for MSWord™ users who care about about information leakage. This result might also be of importance to organizations and individuals that comply with regulations on data leakage practices such as the Health Insurance Portability & Accountability Act.

The conditions that have allowed this state of affairs to come about are a combination of the MSWord™ program default settings and functionality, patterns of document usage and the opaque and proprietary nature of the document format. The existence of the search engine indices is not strictly required as custom crawlers could perform that element of the exploitation, but they make the job significantly easier and faster. This fact hints at the dangers of any oracle, whether it be of the classical prophetic or modern digital variety. Specifically, if you ask a potentially nefarious question you will get the relevant answer. The urge to publish that drives many people and organizations to (needlessly) put their content on the web also contributes.

We believe these findings strongly motivate education of MSWord™ users though corporate security policies and mainstream exposure in order to combat this problem. The choice of suitable policies and technical measures to combat this effect is not simple, but we have examined two alternatives.

9 Acknowledgments

Our thanks to Dave Kormann and Eric Cronin for helpful discussion during this work and Bruce Schneier for suggesting that we write it down.

10 References

[1] Aviel Rubin and Steve Bellovin. Private communication, November 2002.

[2] Simon Byers, Aviel Rubin and David Kormann. Defending against a denial of service attack against the physical world. In *Workshop of Privacy in Electronic Society*, July 2002.