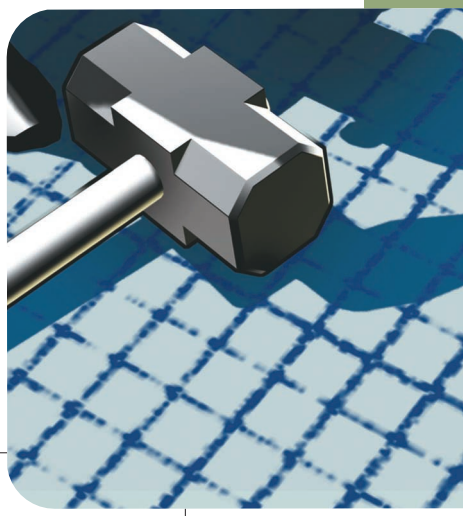
Information Leakage Caused by Hidden Data in Published Documents

This article demonstrates mining for hidden text in published data and concludes that user behavior—in combination with default program settings—creates an uncomfortable state of affairs for Microsoft Word users concerned about information security. The author also presents some countermeasures.



Simon Byers
AT&T Labs-Research

It is well known among the more technical computer user communities that documents written and stored in the Microsoft Word document format might contain hidden data. Some communities of regular computer users are also aware of this fact—for example, people working in the legal sector. However, awareness of this problem is not sufficiently widespread, especially among nontechnical computer users. Such people typically use computers at work and at home but are not computer professionals; thus, they are probably not aware of the dangers that opaque proprietary file formats can carry in their hidden payloads.

This article demonstrates how to automate and scale the search for this hidden text effect. The tools we use are simple and do not require specialized tuning or highly skilled input. We have (and assume) no knowledge of the Word document format itself, reinforcing the simplicity of the techniques discussed. We'll also consider important but commonly neglected behavior patterns in the context of this problem.

Hidden text

Based on anecdotal reports from private conversations and forums across the Internet, we might find the following types of hidden text in a typical Word document:

- names and usernames of the document's creators and their collaborators,
- organizational information of the users involved,
- text specific to the Word program version and document format,
- pathnames of the document in the operating system on which the document was composed,
- information about the hardware on which the document was composed,
- printer names and information,
- email headers or Web server information,
- text deleted from the document at some point prior to the save, and
- text from completely unrelated documents that is only present due to a bug in Word (as opposed to proclivities of the format itself).

We can clarify the last item by referring to an incident that Aviel Rubin and Steve Bellovin told us about in November 2002. The person they described had two separate and unrelated Word documents open at once using MSOffice 97 on MSWindows 98. He then saved and mailed one of these documents to a mailing list; everyone on the list then read the document with his or her usual Word document viewer. One of the mailing list members noticed (by using a non-Microsoft document viewer) that the second document was actually hidden inside the file along with the intended visible one—in fact, he only saw the hidden one initially. Although we don't know how often this happens, this type of occurrence can be more catastrophic than most conventional hidden data mechanisms in terms of data leakage.

Two well-documented public examples illustrate isolated cases of information leaking through hidden text mechanisms. In February 2003, the UK government issued a report in Word format—ultimately known as the "Dodgy Dossier"—on the alleged existence of weapons of mass destruction (WMDs) in Iraq. Some consternation arose when the names of various editors of the document were found hidden inside the file. Ultimately, the

roles of these four individuals were called into question in connection with doubts about the quality of British intelligence before the second Iraq war (see www.computer bytesman.com/privacy/blair.htm). In the late 1990s, Kenneth Starr issued a report in WordPerfect format about US President Bill Clinton's involvement with White House intern Monica Lewinsky. Quirks in the format conversion process brought to light footnotes in the report that ostensibly had been deleted (see http://catless.ncl.ac.uk/Risks/19.97.html#subj3).

The second document was actually hidden inside the file along with the intended visible one.

Tools and implementation

In the methods we describe in this article, we use some commonly available tools:

- Antiword (www.winfield.demon.nl/) is a free, open-source, command-line tool that turns a Word document into plain ASCII text.
- Catdoc (www.45.free.net/~vitus/ice/catdoc/) is similar to antiword in most respects, but sometimes gives slightly different results. To view Word documents sent to them by point-and-click-oriented colleagues, Unix users routinely use both these tools.
- "Strings" is a standard tool that parses the data in any file and extracts those segments that are printable strings of regular text characters.
- Perl is a script programming language useful for manipulating data.

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

Finding content

The first task is to automatically find content to examine. Our predominant source is the open Internet. To find documents on the Internet, we could construct a custom crawler or use the results of existing crawlers; we pick the latter approach and simply use a mechanized interface to any of a number of available search engines.[1] The result of this search is a list of URLs that each gives the address of a Word document on the open Internet.

For search terms, we simply use lists of potentially relevant words. A general data-gathering exercise would employ random word lists whereas we could construct targeted lists for document searches specific to a particular industry or subject type. Example sectors of special interest might include military, government, health care, education, and corporate. Each of these can be targeted effectively, albeit with some small amount of error, by use of appropriate keywords.

Data acquisition

For each URL found with the previous step, we simply download its associated Word document to disk. We store it and all its known originations indexed by the document's MD5 hash value in a regular file system tree. We then invert this mapping for use in searches of content based on origination. This technique is a routine part of many scaled, open Internet, data-gathering exercises.

We also can use search-result extension techniques. One example is to examine all returned URLs for a directory path component such as /docs/. We then attempt to get the virtual directory listing and all relevant documents at that location. This can be a very effective method of targeting content types on the Internet, typically expanding results by 20 percent or more.

Hidden data extraction

Once we've amassed a sizable collection of documents, we can apply various methods for finding hidden text automatically. We use a variety of 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 is completely reliable in parsing Word documents, so we accept some noise in our findings. Of course, such noise is to be expected from an analysis that does not use inside knowledge of hidden data storage, and it's of little consequence because any specific example that seems to warrant further scrutiny will be verified more carefully by hand.

When tables are present, catdoc is generally easier to work with, but catdoc seems to fail slightly more often than antiword in general parsing. Using this *reference parse*, we can see what each option outputs:

- *Antiword method*. We compare the reference parse with a parse taken with the –s option in antiword. (The –s option explicitly tries to display hidden data.)
- *String-matching method*. 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 not found in the reference parse is deemed interesting.
- *Word-matching method*. For each word in each string in the strings output, tabulate the number of occur-

rences of that word in the strings output and in the reference parse. Any word with differing counts in the two is deemed potentially interesting—any word that has zero occurrences in the reference parse is even more so.

We now can apply some simple postprocessing to the results of these heuristics or combinations of them.

Heuristics and results

We first examine the type of output the three previously described methods yield:

- The antiword method rarely finds hidden text and failed on our own canonical examples. The changes it finds can be very small (such as a punctuation change) or as large as several deleted lines. This technique does not reveal usernames, pathnames, and other metadata.
- The string-matching method yields several kinds of results. Some are short junk strings, which we can discount. Username, pathname, printer, and email data is exposed in this way in such a manner as to make it instantly recognizable. This method also returns deleted words or lines containing a deleted or altered word. On Word documents without tables or too much exotic structure, both the catdoc and antiword reference parses return the same results without much effort.
- The word-matching method yields some information about the Word document's internal structure as well as deleted text. Structural information can consist of two similar copies of the document text that differ in some small way. 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.

We intentionally restricted our methods to very simple analysis techniques for two reasons. First, we want to demonstrate how easy this is to implement, and second, to demystify this work so that regular Word users can understand the problem and personally assess their risk. However, custom Word document disassemblers could easily be written from scratch or from the source code of existing utilities such as catdoc, antiword, or OpenOffice. Doing this requires some knowledge of the document format or some reverse-engineering expertise, skills that many individuals possess. We also note the possibility of connecting to a Windows server and having Word automatically convert documents to an open format, but such a conversion is generally not robust or fast enough for use on the scales that we find valuable.

In our experiment, we obtained Word documents at a rate of about 1,000 per hour through a regular cable modem using no parallelization. We spent substantial amounts of the elapsed time in connection timeout events, so the User Agent timeout should be set quite short, with only one or two retries attempted. The first 100,000 documents occupied 16 Gbytes of space.

Some of the retrievals resulted in a file that was either corrupted or not apparently a Word document: we ignored these. All valid Word documents retrieved had some hidden text in them, as the word- and string-matching methods found, although some portion only had the expected Word-related strings and simple metadata. As mentioned, applying the antiword method rarely produced results—in our tests, in only 75 of the first 100,000 documents.

Overall, counting words found by the word-matching method, about half of the documents examined contained between 10 and 50 hidden words, a third between 50 and 500 words, and around 10 percent had more than 500.

Here's an innocuous but real example that illustrates our findings; it happens to be the first document we processed during initial code testing with the string-matching method. All names and organizations have been changed to protect the innocent:

```
1|/m/A
3|/o=Fake University/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
5
1|yHYL
```

The word-matching method yields some information about the Word document's internal structure as well as deleted text.

We include the (sanitized) raw results before post-processing to show how usable the returned results are. 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. This notification is a benign illustration of how the most dangerous examples of leakage might occur—for instance, in text specifically deleted shortly before publication, yet it remains hidden within.

The antiword method produced nothing when applied to this example, but the word-matching method reproduced the same results in a slightly different manner. This particular example has a relatively large amount of the unimportant small junk strings.

The shortest results from applying the string-matching method returned about eight strings. Here's an example, again with the names 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. The word-matching method produced substantially the same results on this example with a couple of extra junk strings.

Use in higher-level analyses

You can use these techniques as building blocks for higher-level analyses of data that might have important end goals with financial and personal consequences. We can think of two examples, one benign and one less so, that illustrate this.

Identity theft

Examining the Word documents in our corpus that appear to be resumes and checking for deleted Social Security Numbers (SSNs) hidden in the file is extremely simple. An individual might include an SSN in the files he or she sends to prospective employers but delete it from the version put online to guard against identify theft. It's even possible to use a keyword in the initial document search to target resumes in particular.

Plagiarism detection

Using the methods we described earlier, gathering information on whether a document has been cloned or plagiarized from another document is possible. 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.

Recommended responses to this problem

In considering responses to this effect, let's look at a realistic scenario involving Word documents:

- Alice sends a mission-critical memorandum written in Word to Bob, who might be in a different organization or corporation—for example, he could be 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 the 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 Word users behave as Bob does in our example, so what's the best way to avoid this kind of problem?

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

One such measure might be to force all personnel in an organization to use a Word document scrubber,

which 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, mainly in the legal profession. A primary problem with this procedure is enforcing its adoption: as with all policy-mandated security procedures, if it involves an extra step, it might not get used unless technical measures assist with compliance.

Another problem with this approach is that even if Alice scrubs her outgoing Word content, she can't 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. Even if Alice sends Bob plaintext, there's still the chance he will convert it into a Word document and end up inadvertently releasing the data.

Some organizations already have specific recommendations on the use of Word: to expose and then remove hidden text during general practice, they use nothing but various options and actions in Word itself (see www.mltweb.com/prof/testdoc.doc). Using our experimental results described earlier, the string-matching method outputs this:

```
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 our methods don't find in this case is the text the author purposefully hid, so it's potentially of less interest because the author is aware of its existence.

Clearly, the specification of a corporate security policy is somewhat difficult. As part of our everyday work, this problem has shown itself to be important: our employer, AT&T, has thousands of Word documents, large amounts of proprietary data, large numbers of industry collaborators, and extensive material on public Web sites. The collaboration of several suborganizations must be secured for any useful impact to be made in preventing leakage through this channel.

The conditions that have allowed this state of affairs to come about are a combination of Word's default settings and functionality, document usage patterns, and the document format's opaque and proprietary nature. Search engine indices are not strictly required—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 is 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, which drives many people and organizations to (needlessly) put their content on the Web in the first place, also contributes.

We believe our findings strongly motivate further education for Word users, either through corporate security policies or mainstream exposure. The choice of suitable policies and technical measures to combat this effect is not simple, but the two alternatives we described here are a good start. This article is specific to Word, but our techniques could be extended to deal with other file formats where such opportunity for hidden data exists. □

Acknowledgments

Thanks to Dave Kormann and Eric Cronin for helpful discussion during this work and Bruce Schneier for suggesting the discussion be written down. Two reviewers also contributed useful suggestions on style and content.

Reference

1. T. Calishain and R. Dornfest, *Google Hacks*, O'Reilly, 2003.

Simple everyday actions can inadvertantly place sensitive data squarely in the public domain.

Simon Byers *is Senior Member of Technical Staff at AT&T Labs Research. His principle interests are data security, personal privacy, acquisition and processing of semistructured data, and the potential for abuse in emerging complex information oriented systems. He has a PhD in statistics from the University of Washington. Contact him at byers@research.att.com.*