A Non-Algorithmic File-Type Independent Method for Hiding Persistent Data in Files

Maha Sabir[1], James Jones [2], Hang Liu[1]
[1]Electrical Engineering and Computer Science
CUA, Washington, DC, USA,
{65sabir, liuh}@cua.edu
[2]Department of Computer Forensics
GMU, Fairfax, VA, USA,
jjonesu@gmu.edu

Abstract

Digital content is most often stored in files, which may be thought of as structured containers for data. This structure facilitates the processing and rendering of the data for human or machine consumption, and also enables the storage of metadata related to the stored content. A side effect of this structured container approach is that the stored file contains more information, sometimes much more, than the actual data that is rendered or available to the receiving human or machine. Additionally, these structures have gaps and other areas where additional data may be stored, unknown to the file owner or subsequent processor. In this paper we propose and test a non-algorithmic and file-type independent approach for hiding persistent and stealthy data in files. This approach may be used to surreptitiously tag files for attribution or tracing purposes, as well as to search for data hidden in existing files. Our approach is not algorithmic like steganography and cryptography. Rather, we take a black box approach to find candidate hiding locations, then we test each of these locations for file integrity and persistence. For our tests, we hid data in MS Word documents using the Office Open Extensible Markup Language (OOXML) format, although our work easily generalizes to other formats. We found multiple locations which allowed for the persistent and benign storage of additional data under various usage scenarios. The main contributions of this paper are: a methodology for identifying conditions favorable for hiding benign and persistent data in arbitrary file types, a methodology for testing these conditions, and empirical results using OOXML formatted files.

Keywords: Digital Forensics, Stealth Watermarking, Anti-Forensics, Data Hiding, File Dead Space, OOXML

1. Introduction

Digital data exists in many forms, at rest and in transit. Digital data is commonly stored or transmitted in the form of files, which provide a format for, and wrapper around, the stored data. Different file types have known structures, which allow users and programs to store data in a particular format that a subsequent user or program will be able to process. These structures may also contain metadata about the data stored in the file, which may also be used by subsequent users or programs. The examination of digital data to determine past events is the domain of digital forensics. Digital forensic examiners usually analyze data at rest and occasionally in transit, and are interested in the contents and provenance of digital data in all of its forms.

Digital forensics experts use a scientific approach for investigating computer crimes and this involves methods for preserving, collecting, validating, identifying, analyzing, documenting, and presenting digital evidence to prosecute criminals (Ademu, Imafidon, & Preston, 2011). On the other side, criminals are using anti-forensic techniques like hiding, altering, and destroying data to elude and forestall forensic investigations (Jain & Chhabra, 2014).

Anti-forensic techniques are broadly categorized into three categories: data destruction, trail obfuscation, and data hiding (Beer et al., 2015; Jain & Chhabra, 2014; Kessler, 2007, p. 4). Data destruction involves wiping data from a digital storage device. Cybercriminals use this technique to destroy incriminating evidence using native or customized tools. When digital evidence is deleted from a storage device using a simple delete process, it can often be retrieved by a forensic expert because the data remains intact on the media (Beer et al., 2015). Data destruction overwrites the data on the media to prevent recovery. Trail obfuscation involves manipulating and altering system and file metadata such as creation, modification, and access timestamps, email headers, and log files (Beer et al., 2015; Jain & Chhabra, 2014; Kessler, 2007). As with data destruction, trail hiding may be accomplished using native or customized tools. Data hiding is an anti-forensic technique for storing data in places where it is not likely to be found or rendered comprehensible. This is achieved using either algorithmic or non-algorithmic data hiding techniques. There are two algorithmic data hiding techniques: steganography and cryptography. Steganography uses algorithms to hide data within other files, while cryptography scrambles the data using keyed manipulation to make it unintelligible to anyone without the appropriate decryption key (Raggo & Hosmer, 2013). Alternatively, in non algorithmic data hiding techniques, data is hidden in files and storage devices without the use of algorithms. This technique is useful for watermarking digital evidence and classified documents for attribution and traceability purposes. Both non-algorithmic hiding and steganography are also used for the covert storage and transmission of information.

The main objective of this paper is to study the persistence and impact of inserting character strings into apparently unused file regions. We use a non-algorithmic and black box approach to identify candidate file locations, where such locations are empirically determined *structural dead space* in the file. By comparison, techniques like steganography use algorithms to hide files within other files such as images by altering low order bits of pixels, cryptography uses algorithms to scramble data deterministically, and other hiding techniques require knowledge of the file structure.

The main research question posed in this paper is: *Can we hide data in files that is persistent, benign, recoverable and stealthy?* To answer this research question, the paper addresses two supporting research questions: 1) which tags will survive and under what conditions? 2) Are there patterns that determine which tags survive?

This research has two primary practical applications. First, cyber security practitioners, forensic examiners, and investigators can tag digital data to trace and attribute leakage of sensitive, classified, or otherwise interesting digital content. Second, researchers and investigators can develop techniques to find stealthy data in existing digital content, and can develop methods for the sanitization of files suspected of containing stealthy content.

The rest of the paper is organized as follows: in Section 2 we review related work on data hiding and anti-forensics, in Section 3 we present our research methodology and description of the data, Section 4 contains results and discussion, and Section 5 presents our conclusions and future work.

2. Related Works

Most prior work has focused on hiding data in specific file formats such as the Open Extensible Mark-up Language (OOXML) and Open Document Format (ODF). Both of these formats are XML-based making them portable compared to the earlier binary versions. In this section we first describe the OOXML structure and then review related research on hiding data in these and other file types.

2.1 Microsoft Word OOXML File Format

Prior work by Park, Park, & Lee (2009) studies data hiding in the current Microsoft Office document format called Office Open Extensible Mark-up Language (OOXML). Documents in the OOXML format are stored in a file zip archive comprising several components: *content type*, *relationship*, and *common*. The relationship components contain information about the zip file archive package and its parts. The *common* parts component is where the main document body with most printable characters, documents styles, and document formatting information are stored, while the *content type* component defines the

content types or parts inside the file zip archive. There is also a *properties* component which tracks growth or shrinkage of a file as data is added or removed from a document (Fu, Sun, & Xi, 2015).

2.2 Hiding Data in Files

There is limited research on hiding persistent data in files non-algorithmically and independent of file type. Early research was conducted on hiding data in files like web pages and an early binary version of Microsoft office document files (Cantrell & Dampier, 2004). In that work, data was hidden in file dead space, which was considered as any space with at least two hexadecimal zeroes, using a steganography application. Since this paper was hiding data in all available file dead space, and the definition of dead space was overly broad, some documents got destroyed. The major recommendation from that paper expressed the need for further research on hiding data in Microsoft Word binary files. Our work differs from this study in that we seek out larger contiguous regions of likely dead and inert space, we test each location individually, and we study the patterns of apparent dead space in order to generalize our approach and increase its robustness.

Other research on data hiding uses earlier binary versions of Microsoft Word's opaque format called Microsoft Compound Document File Format (MCDFF) to extract hidden meta-data. This work also proposes steganographic tools to hide encrypted data (Castiglione, De Santis, & Soriente, 2007). That research uses steganography to read meta-data for objects embedded in Microsoft Word using Object Linking and Embedding (OLE) and then hides encrypted data in structured storage spaces of the office application. Comparatively, this work requires knowledge of the file structure, while our approach uses empirically determined file dead space and is not dependent on file type.

Another early paper applied steganography to hide data using document tracking techniques in Microsoft Office 2003 (Liu & Tsai, 2007). According to that paper, the researcher's proposed a method for storing hidden data in documents using stenography and document tracking tools. That paper uses a binary version of Microsoft office documents and document tracking tools, which is quite different from our approach.

Other work by Park, Park, and Lee (2009) conceals data in Microsoft Office 2007 files and proposes an algorithm and tool for detecting such data. According to that paper, data is hidden as XML comments in related parts of the XML schema that forms the OOXML file zip archive yet this schema does not support comments. As with other prior work, this research requires knowledge of, and leverages, the file structure. We treat the files as a black box and hide data within empirically determined file dead spaces.

Work by Park and Lee (2009) presents a forensic examination of binary versions of Microsoft Office PowerPoint 1997-2003 files. That work used metadata generated when contents of documents were deleted or edited (Park & Lee, 2009). According to that paper, the data was obtained from the "*Allow fast saves*" feature in Microsoft PowerPoint documents, which generates residual information as well as identifiers like *SlideIDs* and *ObjectIDs,* hence revealing data relationships that can be used to determine information about the content.

Similarly, work by Garfinkel and Migletz (2009) evaluated forensic implications of XML-based files for office document formats like OOXML for Microsoft and Open Document Format (ODF) for Open Office. The paper identified several attributes for hiding data in XML-based document formats: altering revision to change a document's editing history without enabling tools to track document changes, altering timestamps within XML files regarding when the documents are created or modified, encrypting data in content parts of the zip file archive, and hiding data in XML comments. As noted previously, such techniques leverage the file structure and format while we do not.

A recent study by Castiglione et al., (2011) investigated hiding data in OOXML file zip archives and evaluated steganographic methods like altering zip compression algorithm, office macros, zero dimension image, and revision identifier values. The paper first attempted to alter Microsoft's default zip compression algorithm to hide binary data. The paper then investigated data hiding using macros whereby data is hidden either as comments or variables within the macro code of the OOXML document without changing the extension of the document to say *.docm* which is a Microsoft OOXML document

with macro functions. Furthermore, data was also hidden using zero image dimensions in OLE-object properties like setting image dimensions to zero, positioning the image in the upper-left position of the document page and placing it behind text. Lastly that paper proposed hiding data by the revision value identifier (*rsid)*, which tracks the editing session of a document, whereby values of *rsid* attributes are replaced with codified hexadecimal data.

Other work proposed several methods and a forensic tool to detect covert communication in Microsoft Office 2007-2013 documents (Fu et al., 2015). The paper detected hidden data in XML files of the OOXML zip file archive in the form of extra fields in*document.xml* use of codified data to replace values of attributes for revision identifiers, changing macros variables and adding comments, using zero dimension images, and adding binary data to zip compression algorithms. The paper subjected these data hiding approaches to five conditions: clear format, save as, copy, modify, and delete.

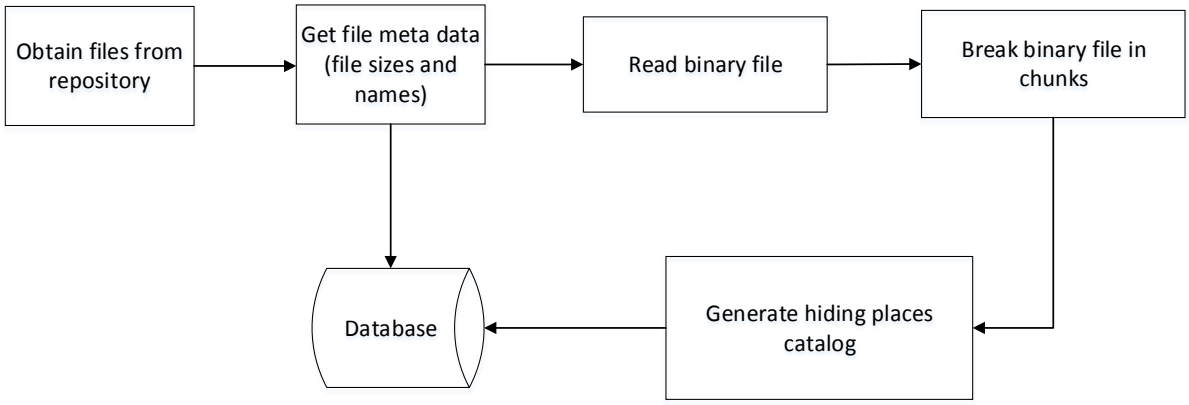
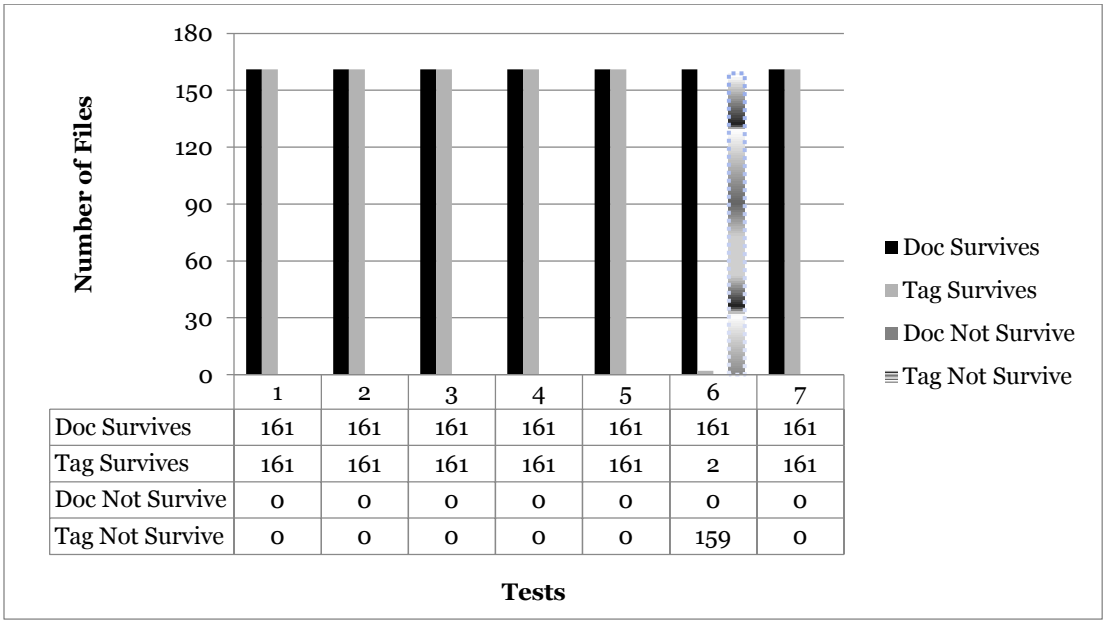


Figure 1: Process For Finding Hiding Places in Files



	1	2	3	4	5	6	7
Doc Survives	161	161	161	161	161	161	161
Tag Survives	161	161	161	161	161	2	161
Doc Not Survive	0	0	0	0	0	0	0
Tag Not Survive	0	0	0	0	0	159	0

Figure 2: Document And Tag Survivability

3. Methodology & Experiment Design

3.1 Dataset Description

We used a publicly available dataset of government documents called GovDocs (Garfinkel, Farrell, Roussev, & Dinolt, 2009). The documents are available online at https://digitalcorpora.org/corpora/govdocs. In this dataset we collected all of the 161 Microsoft DOCX files in the corpus. We used Microsoft Word 2007 to conduct these experiments, and experiments with other versions of Word are ongoing.

The files in our dataset comprised documents containing plain text, document templates with Visual Basic for Application (VBA) components, and plain text with OLE objects.
We define a *file dead space* as a string of 16 or more consecutive 0x00 (hex 00) values.

Measure	**File size (KB)**	**File dead spaces(count)**
Maximum	9,495	153
Minimum	10	5
Average	210	8
Median	30.32	5.00
Standard Deviation	849.38	12.25
Correlation (R)	*0.318*	

Table 1: Descriptive Statistics of the Dataset

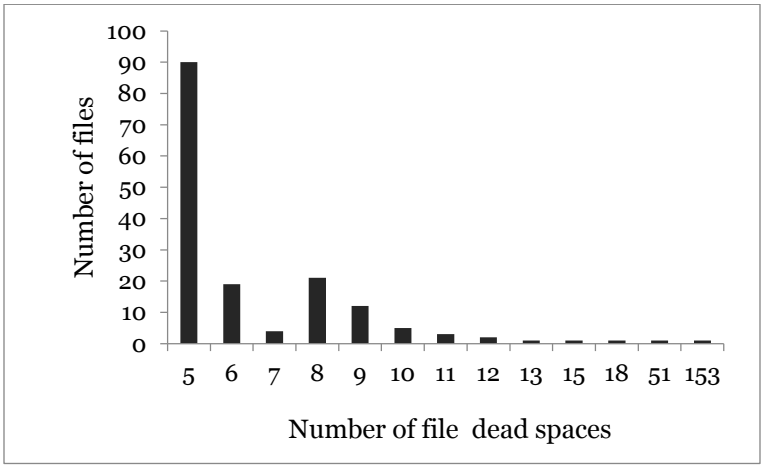


Figure 3: Histogram for File Dead Spaces

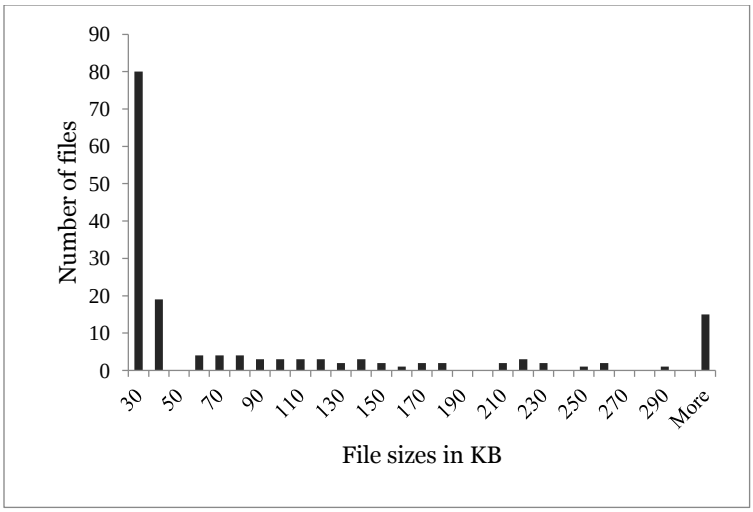


Figure 4: Histogram for File Sizes

Table 1 is a statistical summary of the dataset. The maximum and minimum sizes of the files in the dataset are 9,495 KB and 10 KB respectively while the average file size of the documents in the dataset is 210 KB. The average frequency of file dead spaces for documents in the dataset is 8 (e.g., for all files in our dataset, we found an average of 8 dead spaces per file in a range of 5-153). The maximum and minimum frequency of file dead spaces is 153 and 5 respectively. The median file size and frequency of file dead spaces for the dataset is 30.32 and 5 respectively while the standard deviations for file size and file dead space frequency is 849.38 and 12.25 respectively.

Figure 3 and Figure 4 show the distribution of numbers of file dead spaces and file sizes in our dataset of documents. Kurtosis for file size and file dead space frequency distribution is 91.74 and 126.79 respectively, while skewness for file size and file dead space frequency distribution is 8.91 and 10.87 respectively. The peaks for distribution curves for file size and file dead space frequency are both left-leaning hence the positive kurtosis. Means greater than the medians indicates positive skewness in the distribution of the data (Levine & Stephan, 2014). Table 1 illustrates that the means (or averages) for file sizes and file dead spaces are greater than their medians. This confirms that the distribution for file sizes and file dead spaces is positively skewed as shown in Figure 3 and Figure 4.

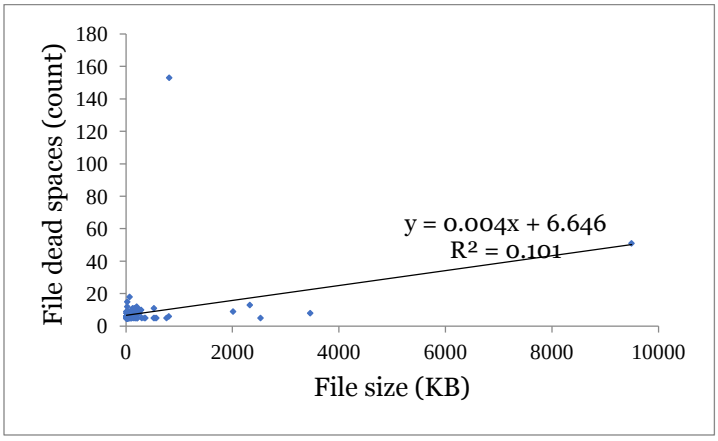


Figure 5: Scatter plot for File size and File dead spaces with outliers

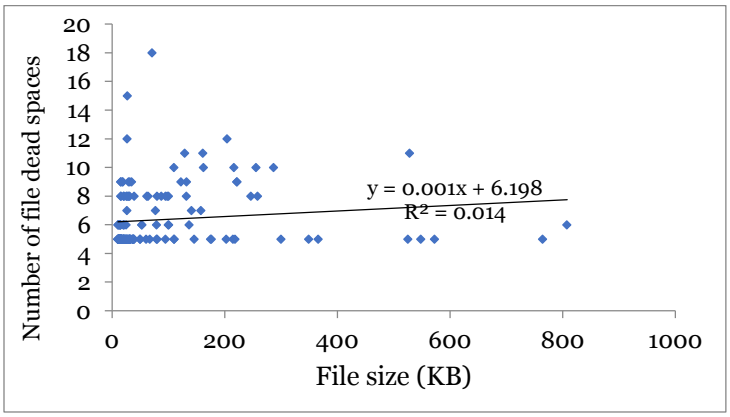


Figure 6: Scatter plot for File size and File dead spaces without outliers

3.2 Finding Hiding Places in Files

Figure 3 is the histogram for file dead spaces for the documents in our dataset. We observed the anomalous values for dead space counts of 6 and 7 (the rest of the data fits a nice smooth exponential curve). This could be attributed to the specific files in our dataset, or may indicate an underlying causal factor which we have yet to reveal. In the same histogram, 90 documents had 5 file dead spaces which was the highest frequency in the dataset. There were also 21 documents with 8 dead spaces, 19 documents with 6 dead spaces, 12 documents with 9 dead spaces, 5 documents with 10 dead spaces, 4 documents with 7 dead spaces, 3 documents with 11 dead spaces, 2 documents with 12 dead spaces, and the remaining 5 documents had 13 to 153 dead spaces (a long, thin tail).

Figure 4 is a histogram for file sizes of the dataset. Most of the files (80) in our dataset are in a file size range 10KB and 30KB. There are 19 files within a file size range of 30KB and 40KB while 15 files are more than 300KB. The file sizes for rest of the files are distributed between 60KB and 290KB.

The correlation coefficient of 0.32 in Table1 reveals that there is some correlation between file size and the frequency of the file dead spaces in the dataset. The scatter plot in Figure 5 and 6 reveal the nature of the relationship that exists between these two variables. Figure 5 shows that most of the data points have less than 20 file dead spaces and are of size less than 2000 KB. We removed the outliers from the scatter plot in Figure 6, revealing that most of these files have between 5 and 10 file dead spaces inclusive and there is very little correlation between the number of file dead spaces and file size, although the higher spread at lower file sizes is interesting but as yet unexplored.

In Figure 1, we outline our dead space discovery process. We obtain and process files in the dataset from a repository. To process these files we obtain and store metadata like file size, file name, etc., for each file, and then we read the file as a raw binary object. To process the files for file dead spaces, we are looking for 16 consecutive hexadecimal zeros, which constitute a dead space for our experiments. We store the location and size of these dead spaces in the catalog with the associated file metadata.

Test	Test Name
1	Copy on device
2	Copy off device
3	Open/Close/No Modify/No Save
4	Open/Save/Close/No Modify
5	Open/Modify/Close/No Save
6	Open/Modify/Save/Close
7	Open/Modify/Terminate

Table 2: Description of the Tests

3.3 Tagging Files and Survivability Testing

To tag the files we use a 16 byte string which is written to all the dead spaces of each file, collectively and separately. For dead spaces greater than 16 bytes, we place a tag both at the beginning and middle of the dead space. We previously observed that tags at the end of a dead space never survived unless a tag at the beginning or middle also survived.

We also observed that the first three dead spaces at [*Content_Type].xml, _rel/.rels,* and *word/_rel/document.xml.rels* are stable and do not change when we execute delete, write and save operations to modify a document. The tagging was automated using python and the tags were manually verified using a graphical hex editor like *WinHex*.

For survivability testing, we determine whether the tags survive when subjected to specific conditions and also whether the tags destroy the files under the same conditions. In Table 2, we conducted seven survivability tests on each tagged file:*Test 1* is copy file (on device); *Test 2* is copy file (off device); *Test 3* is open, close (no modify, no save); *Test 4* is open, save, close (no modify); *Test 5* is open, modify, close (no save); *Test 6* is open, modify, save and close; and *Test 7* is open, modify and terminate. The operations for these tests are designed to cover a range of possible actions in order to identify the actions which are likely to affect, or not affect, tag persistence and document integrity.

4. Results And Discussion

A summary of our findings is shown in Figure 2. All the tests were automated and the results were manually verified. We verified the files from both the automated and manual tests by comparing them byte for byte to ensure consistency of the results.

In Test 1, we copied the tagged files from one folder to another on the same drive. In this test all the tags in the 161 files persisted and also the tagged documents could be opened and closed without any error messages.

In Test 2, we copied the tagged files off the device to another file server. In this test all the tags as well as the tagged documents survived.

Test 3, we opened the tagged documents, closed the files without making any modifications or even saving any changes. In that test all the tags survived and also the tagged documents were not destroyed.

In Test 4, we opened the tagged documents, saved the documents without making any modifications, and closed the files. We observed that all the tags and the documents survived after running this test.

In Test 5, we opened the tagged documents, made changes by adding a plain text string and then closed the files without saving any changes. In this test we observed that all the tags and the tagged documents survived.

For Test 6, we opened the tagged documents, made modifications to the files by adding a string, saved the changes and then closed the files. We observed that only two tags survived and159 tags did not survive. However, all the 161 tagged documents remained viable. The two tags that survived were in template files with only OLE and VBA objects and a limited number of dead space locations.

Lastly, in Test 7 we opened, modified and terminated the tagged documents. In this test we observed that both the tagged documents and the tags survived.

The tests from the experiments reveal that it is possible to hide persistent data in file dead space of DOCX (OOXML) files provided the documents are not modified and the changes are not saved. We observed that tags in zip file archive files in OOXML format do not survive when such documents are modified and saved. This was also similar to findings made in a early paper on hiding steganographic data in an older binary version of Microsoft Word documents (Cantrell & Dampier, 2004). All the additional experiments we conducted reveal that hidden data in file dead space persists when operations like opening, closing, terminating, copying and saving without making modifications are performed on the documents. These tests indicate conditions that are conducive for survivability of persistent hidden data in files.

Another finding in this paper is that files have several internal dead spaces. However, persistent data could only be hidden in the first three file dead spaces of the documents which had 16 byte hexadecimal zeroes when subjected to the conditions described above. When we attempted to tag all the dead spaces of 16 byte hexadecimal zeroes as well as the file dead spaces which had less than 16 byte hexadecimal zeroes, the tags either disappeared, or the tagged documents were corrupted such that documents displayed error messages when opened.

5. Conclusions & Future Work

As digital content creation and storage continues to increase, it is important to understand conditions that allow adding stealthy and persistent data to files. This is particularly useful for data protectors, investigators, and forensic examiners to trace and attribute large volumes of digital evidence in the form of files. Similarly, the work presented here can form the foundation for techniques to find and eradicate surreptitiously hidden data.

We determined that it is possible to empirically find locations suitable for storing data without knowledge of the files internal structure, and that these locations can survive many but not all operations. In particular, editing a file using the associated application proved destructive to inserted data, while other operations appeared to have no effect on the tag or the viability of the tagged document.

For Future work we will expand the suite of applications in our tests, apply the approach to other file formats and applications, and further analyze the characteristics of dead space suitable for persistent and benign tagging.

References

Ademu, I. O., Imafidon, C. O., & Preston, D. S. (2011). A new approach of digital forensic model for digital forensic investigation. *International Journal of Advanced Computer Science and Applications*, *2*(12), 175–178.

Beer, R. de., Stander, A., & Belle, J. (2015). Anti-Forensics: A Practitioner Perspective. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, *4*(2), 390–403.

Cantrell, G., & Dampier, D. D. (2004). Experiments in hiding data inside the file structure of common office documents: a stegonography application. In *Proceedings of the 2004 international Symposium on information and Communication Technologies* (pp. 146–151). Trinity College Dublin.

Castiglione, A., D'Alessio, B., De Santis, A., & Palmieri, F. (2011). Hiding Information into OOXML Documents: New Steganographic Perspectives. *Journal of Wireless Mobile Networks Ubiquitous Computing and Dependable Applications*, *2*(4), 59–83.

Castiglione, A., De Santis, A., & Soriente, C. (2007). Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. *Journal of Systems and Software*, *80*(5), 750–764. https://doi.org/10.1016/j.jss.2006.07.006

Fu, Z., Sun, X., & Xi, J. (2015). Digital forensics of Microsoft Office 2007-2013 documents to prevent covert communication. *Journal of Communications and Networks*, *17*(5), 525–533. https://doi.org/10.1109/JCN.2015.000091

Garfinkel, S. L., & Migletz, J. J. (2009). New XML-Based Files: Implications for Forensics. *IEEE Security Privacy*. *7*(2), 38–44.

Garfinkel, S., Farrell, P., Roussev, V., & Dinolt, G. (2009). Bringing Science to Digital Forensics with Standardized Forensic Corpora, DFRWS, Montreal, Canada.

Jain, A., & Chhabra, G. S. (2014). Anti-forensics techniques: An analytical review. In *2014 Seventh International Conference on Contemporary Computing (IC3)* (pp. 412–418). https://doi.org/10.1109/IC3.2014.6897209

Kessler, G. C. (2007). Anti-forensics and the digital investigator. In *Proceedings of the 5th Australian Digital Forensics Conference* (p. 1). Mt Lawley, Western Australia, Edith Cowan University.

Levine, D. M., & Stephan, D. F. (2014). Shape of Distributions. In *Even You Can Learn Statistics and Analytics: An Easy to Understand Guide to Statistics and Analytics, Third Edition* (3rd ed.). Pearson FT Press. Retrieved from http://proquest.safaribooksonline.com/book/statistics/9780133382693/chapter-3dot-descriptive-statistics/ch03_html

Liu, T. Y., & Tsai, W. H. (2007). A New Steganographic Method for Data Hiding in Microsoft Word Documents by a Change Tracking Technique. *IEEE Transactions on Information Forensics and Security*, *2*(1), 24–30. https://doi.org/10.1109/TIFS.2006.890310

Park, B., Park, J., & Lee, S. (2009). Data concealment and detection in Microsoft Office 2007 files. *Digital Investigation*, *5*(3–4), 104–114. https://doi.org/10.1016/j.diin.2008.12.001

Park, J., & Lee, S. (2009). Forensic investigation of Microsoft PowerPoint files. *Digital Investigation*, *6*(1–2), 16–24. https://doi.org/10.1016/j.diin.2009.05.001

Raggo, M. T., & Hosmer, C. (2013). Data hiding: exposing concealed data in multimedia, operating systems, mobile devices and network protocols. Waltham, Massachusetts USA: Elsevier.