Digital Investigation xxx (2018) 1–7

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Dismantling OpenPuff PDF steganography

Thomas Sloan*, Julio Hernandez-Castro

University of Kent, School of Computing, Canterbury, CT2 7NF, UK

ARTICLE INFO

Article history:
Received 9 October 2017
Received in revised form
5 February 2018
Accepted 12 March 2018
Available online xxx

Keywords:
Steganography
Steganalysis
OpenPuff
Privacy

ABSTRACT

We present a steganalytic attack against the PDF component of the popular OpenPuff tool. We show that our findings allow us to accurately detect the presence of OpenPuff steganography over the PDF format by using a simple script. OpenPuff is a prominent multi-format and semi-open-source stego-system with a large user base. Because of its popularity, we think our results could potentially have relevant security and privacy implications. The relative simplicity of our attack, paired with its high accuracy and the existence of previous steganalytic findings against this software, warrants major concerns over the real security offered by this steganography tool.

Introduction

Steganography is the process of hiding information in plain sight. This can be used to carry out secret communications or avoid suspicion over the exchange of information. It is important to emphasise that the objectives of steganography are very different to those of cryptography. The latter provides confidentiality but does not disguise the existence of secret data, which is obvious to any observer and can even be automatically detected due to the high entropy of encrypted data. Steganography, however, aims to avoid detection under all circumstances, even when an active warden has full access to all exchanged data. In most cases, steganographic tools and algorithms will also use cryptography to encrypt the contents that will be later hidden, but their objectives are more ambitious; data should be exchanged without raising any suspicion. This ultimately aims for carrier files to be indistinguishable from unmodified, clean sources. Consequently, the main objective of steganalysis, the associated discipline, is the pursuit of identifying and proving the existence of steganography. Additional objectives could be to estimate the size of the hidden contents, and in a few cases fully recover the hidden message and/or key used. The techniques used by the steganalyst will vary, depending on the embedding algorithm that has been used, but will often be statistically themed or system based.

Modern steganography is often aimed at digital media. There is a myriad of tools able to perform steganography over different carrier formats, such as images, audio, video, executable files, games, VoIP, P2P, etc. Any format that contains some form of redundancy can ultimately be employed for steganography (Provos and Honeyman, 2003).

Relevance of modern steganography

Steganography, very much like cryptography, is a dual-use technology. On the one side, it can be used to evade censorship in circumstances where free speech or the free flow of information is limited or restricted. Another use of steganography is to evade the increasingly prying eyes of snooping governments across the world, that try to ban or limit encryption with the excuse that terrorist and other criminals also use it. On the other side, this is partly true as steganography can also help cyber-criminals and terrorists to conceal their communications while exchanging important information and evading prosecution. Steganography is then both a very useful and necessary tool for providing the general public with increased privacy and anonymity, but also can potentially help a very small minority of its users to avoid detection while planning or committing malicious activities. It should be emphasised that steganography tools and its community of users will often have legitimate reasons (i.e. journalists, civil rights and democracy activists under threat of imprisonment, etc.) to use steganography as a means to communicate safely (The Guardian Project, 2017). However, it is unfortunately the criminal use of steganography

* Corresponding author.
E-mail addresses: ts424@kent.ac.uk (T. Sloan), jch27@kent.ac.uk (J. Hernandez-Castro).

https://doi.org/10.1016/j.diin.2018.03.003
1742-2876/

which more often reaches widespread exposure through the media. Recent cases in point are the use of image steganography by a network of Russian spies in the US (Shachtman, 2010), video steganography usage by an Al-Qaeda operative in Germany (Gallagher, 2012), or recent strands of malware employing steganography for communication with their command and control servers (Young, 2015).

In recent years, video steganography has become an increasingly popular technique for data exfiltration. One example is the case of an undisclosed Fortune 500 company that was hit by this type of attack (Paganini, 2017). The use of video steganography allowed for large amounts of sensitive data to be exfiltrated from the company network after the breach, bypassing all the data leakage protection (DLP) tools and intrusion detection systems (IDS) put in place.

Paper organisation

The rest of the paper is organised as follows: We will first introduce the OpenPuff steganography tool, covering some of its features, and discussing its popularity and known previous attacks. We will then focus on its operation over PDF files. Afterwards, we will show how to construct and test an efficient distinguisher capable of detecting its carrier files with high accuracy. Also, we will explore how to estimate the size of the hidden data. We will then provide additional results and insights, obtained after extensively testing our proposed approach. We continue with a discussion of our findings, elaborating on their potential impact. Finally, we close the paper with some comments on the social and moral implications of this research, together with conclusions and ideas for future related works.

OpenPuff

OpenPuff is a tool capable of performing steganography over a large number of different file formats, concretely (as listed on the tool's website) sixteen, covering media such as images (bmp, jpx, pcx, png, and tga formats), audio (aiff, mp3, NEXT/SUN, wav), video (3gp, mp4 mpg, wob) and Flash/Adobe formats (flv, swf, and pdf). The present study is focused on its operation over the PDF format.

OpenPuff description

OpenPuff (currently in version 4.0) is marketed as "Yet **not** another Steganography Software" and is a free semi-open-source tool, which uses the libObfuscate v.2 library, developed by the same author, to perform its cryptography related tasks. The libObfuscate library implements a number of encryption algorithms (including AES, Anubis, Camellia, CAST-256, Clefia, Frog, Hierocrypt3, Idea-NXT, MARS, RC6, Safer+, SC200, Serpent, Speed, Twofish, and Unicorn-A). This large selection of sixteen block ciphers is employed to realise the concept of multi-cryptography, first introduced by its author, where the cipher used to encrypt data is picked pseudo-randomly. It is important to note that, as shown in the architectural design in Fig. 1, these block ciphers are used in the insecure ECB mode, strongly against common practice. It may well be pertinent to add that most of these block ciphers are considered to be very weak or, in some cases, totally broken. Most are not currently in use, and did not even make it to the final stages of the AES competition in 2001.

These design choices are highly non-standard, and some are plainly wrong from a security point of view. We will discuss these issues further in the Conclusions section. libObfuscate also implements four hash functions, all producing a 512-bit digest. These are Grostl, Keccak (the new SHA-3 standard), SHA-2 and Skein. OpenPuff also offers a cryptographically secure pseudo-random number generator (CSPRNG) based on AES but not following any known PRNG standard. It is perhaps relevant to stress that, although libObfuscate's source code is available, OpenPuff is in itself closed source, so we don't know exactly how the steganography algorithms operate. This is again against best practice and can be considered a security issue.

OpenPuff popularity

Despite its highly unusual and sometimes very questionable design criteria, OpenPuff has received notoriously positive feedback and enjoys a very good reputation. Users appear to appreciate its versatility, simplicity, and ability to perform over multiple media. It is quite common to find recommendations and positive reviews for OpenPuff (Zukerman, 2013), which has been short-listed in a number of "Best security tools" or "Best steganography tools" articles, some recently and by prestigious media and cyber-security experts (Hosmer, 2012). Although details are not available on the tool's website, third party sites reveal large download numbers (these figures are as of 27/09/2017), as shown in Table 1. We expect the total numbers of downloads to be significantly higher than those shown below.

Related work

Known steganalytic results against OpenPuff are surprisingly scarce. To the best of our knowledge, much of the related work in this avenue of steganography has been reported in just two academic publications: A recent attack targets the MP4 component (Sloan and Hernandez-Castro, 2015). This work takes advantage of OpenPuff's modification of MP4 flags that are commonly set to a null value. Based on this observation, a script is proposed which, after being extensively tested, demonstrates high detection accuracy. As a result, the MP4 component of OpenPuff should be considered broken and hence totally insecure. The audio component of OpenPuff was examined by Liu et al. (2011) who were able to successfully detect OpenPuff steganography as implemented on an older version of the tool (v3.10). The authors proposed the StegAD scheme to detect audio steganography in cloud services, using an enhanced version of the RS algorithm originally proposed by Fridrich et al (Fridrich and Goljan, 2002).

The use of PDF steganography has been explored both inside and outside academic literature. Several embedding algorithms have been proposed by researchers, while a number of tools also exist for public and commercial use. In current literature, algorithms can either target the PDF format directly, or exploit a form of ASCII steganography, which can then be converted from one text-type format to another. Rafat & Sher (Rafat and Sher, 2013) propose an ASCII based steganographic algorithm robust enough to allow conversion between MS Word and PDF files without losing any of the embedded content.

Alizadeh et al. (Alizadeh-Fahimeh et al., 2012) examine a number of existing techniques for PDF steganography including word/character embedding and the manipulation of incremental updates. However, these are not robust techniques and cannot be used outside of the PDF environment. The authors also examine similar

Please cite this article in press as: Sloan, T., Hernandez-Castro, J., Dismantling OpenPuff PDF steganography, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.03.003

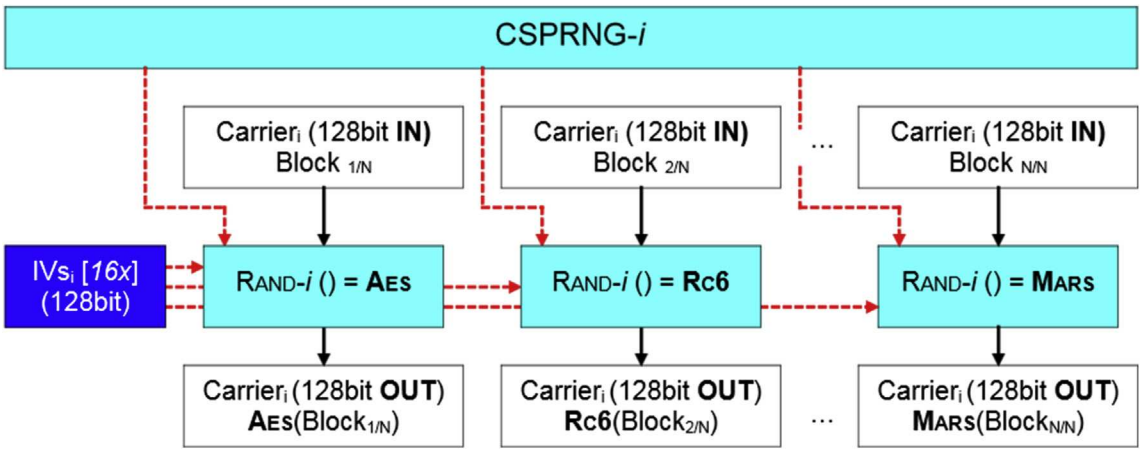

Fig. 1. OpenPuff cryptographic architecture, from (Oliboni, 2017).

Table 1
Approximate download figures for OpenPuff as of 27/09/2017.

Website	Downloads in Last Lear	Total downloads
http://download.cnet.com/OpenPuff/3000-2092_4-75450743.html	2038	8637
http://www.pcworld.com/product/1252470/openpuff.html	0	1420
http://www.softpedia.com/get/Authoring-tools/Authoring-Related/Puff.shtml	808	59,002
http://www.downloadcrew.co.uk/article/27852-openpuff	462	2406
http://www.freewarefiles.com/OpenPuff_program_58719.html	694	5274
http://www.majorgeeks.com/files/details/openpuff.html	1229	10,004
http://www.winpenpack.com/en/download.php?view.913	311	3271
http://www.baixaki.com.br/download/openpuff.htm	494	3499
http://www.brothersoft.com/openpuff-34008.html	13	38,956
http://openpuff.en.uptodown.com/	686	2200
Total	**6735**	**134,669**

but more efficient embedding algorithms such as steganography via hidden PDF components and the manipulation of margins and TJ operators. The authors continue the work of Zhong et al. (2007) and develop two similar algorithms, one with slightly better security but lower capacity, the other with higher capacity but provably worse security.

Furthermore, tools such as StegoStick, DeEgger Embedder, wbStego are examples of the many available to the general public that can perform steganography over PDF files.

On the other hand, there is only a handful of published steganalytic attacks against PDF in the existing literature. One such example is a detection method for word shift steganography in PDF, by Lingjun et al. (2008). The authors propose a blind steganalytic attack against steganography through inter-word space length, through the analysis of the statistical properties of spacing.

OpenPuff PDF steganography

In this section, we present our findings on the operation of OpenPuff over PDF carrier files. For context, we will also include a very brief introduction to the PDF format, highlighting just the basic knowledge required to understand the attack discussed. We will also detail very briefly the state-of-the-art, to put OpenPuff's algorithm into perspective.

The PDF format

The contents of this section are largely based on (Incorporated, 2006; King, 2005) to which we refer any interested readers for further technical info.

The Portable Document Format, known widely as PDF, is a document format published originally by Adobe as a proprietary model in 1993. It was not until 2008 that it was published as an open standard, as ISO 32000-1:2008. The functionality of a PDF file is largely determined by a series of objects such as dictionaries, arrays, streams and other values (character sets, operators, etc) that act as metadata to describe the file. The PDF syntax is best understood by thinking of it in four parts, as described below:

- Objects. A PDF document is a data structure composed from a small set of basic types of data objects.
- File structure. The PDF file structure determines how objects are stored in a PDF file, how they are accessed, and how they are updated. This structure is independent of the semantics of the objects.
- Document structure. The PDF document structure specifies how the basic object types are used to represent components of a PDF document: pages, fonts, annotations, and so forth.
- Content streams. A PDF content stream contains a sequence of instructions describing the appearance of a page or other graphical entities. These instructions, while also represented as

Please cite this article in press as: Sloan, T., Hernandez-Castro, J., Dismantling OpenPuff PDF steganography, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.03.003

objects, are conceptually distinct from the basic objects that represent the document structure.

It may be handy for understanding later sections of this work to become more familiar with how streams operate. In (Incorporated, 2006), we find that a stream object is a sequence of bytes. Furthermore, a stream may be of unlimited length, whereas a string shall be subject to an implementation limit. For this reason, objects with potentially large amounts of data, such as images and page descriptions, shall be represented as streams. A stream consists of a dictionary followed by zero or more bytes bracketed between the keywords **stream** (followed by newline) and **endstream**. The keyword stream that follows the stream dictionary is followed by an end-of-line marker consisting of either a CARRIAGE RETURN and a LINE FEED or just a LINE FEED, and not by a CARRIAGE RETURN alone. The sequence of bytes that make up a stream lie between the end-of-line marker following the stream keyword and the end-stream keyword; the stream dictionary specifies the exact number of bytes. There should be an end-of-line marker after the data and before endstream; this marker shall not be included in the stream length. There shall not be any extra bytes, other than white space, between endstream and endobj. Fig. 2 provides illustrative examples of the PDF format.

Both objects and, in particular, streams play an important role in OpenPuff steganography over PDF files.

Testing and results

In this section, we present the analytic framework that we developed to carry out our successful attack against OpenPuff, and the corresponding results of our analysis. Through our testing method, we analyse the PDF steganography component of OpenPuff. From this, we have constructed a distinguisher and a message estimator (as shown in Appendix 1), to detect the presence of OpenPuff PDF steganography. These tests are discussed and shown in Section Testing and results.

Construction of a distinguisher and estimator

OpenPuff claims that data is pre-encrypted with multi-cryptography before being embedded into a file. The message is then scrambled (shuffled with random indexes), whitened (mixed with a high amount of noise, taken from an independent CSPRNG seeded with hardware entropy) and encoded (with adaptive non-linear encoding, that takes also original carrier bits as input). This stego-system also claims "modified carriers will need much less change and deceive many steganalytic tests (e.g.: chi square test)."

OpenPuff PDF steganography embeds data through the modification of end-of-line (EOL) markers, called 'white-space characters'. In particular, the carriage return (CR) and line feed (LF) characters

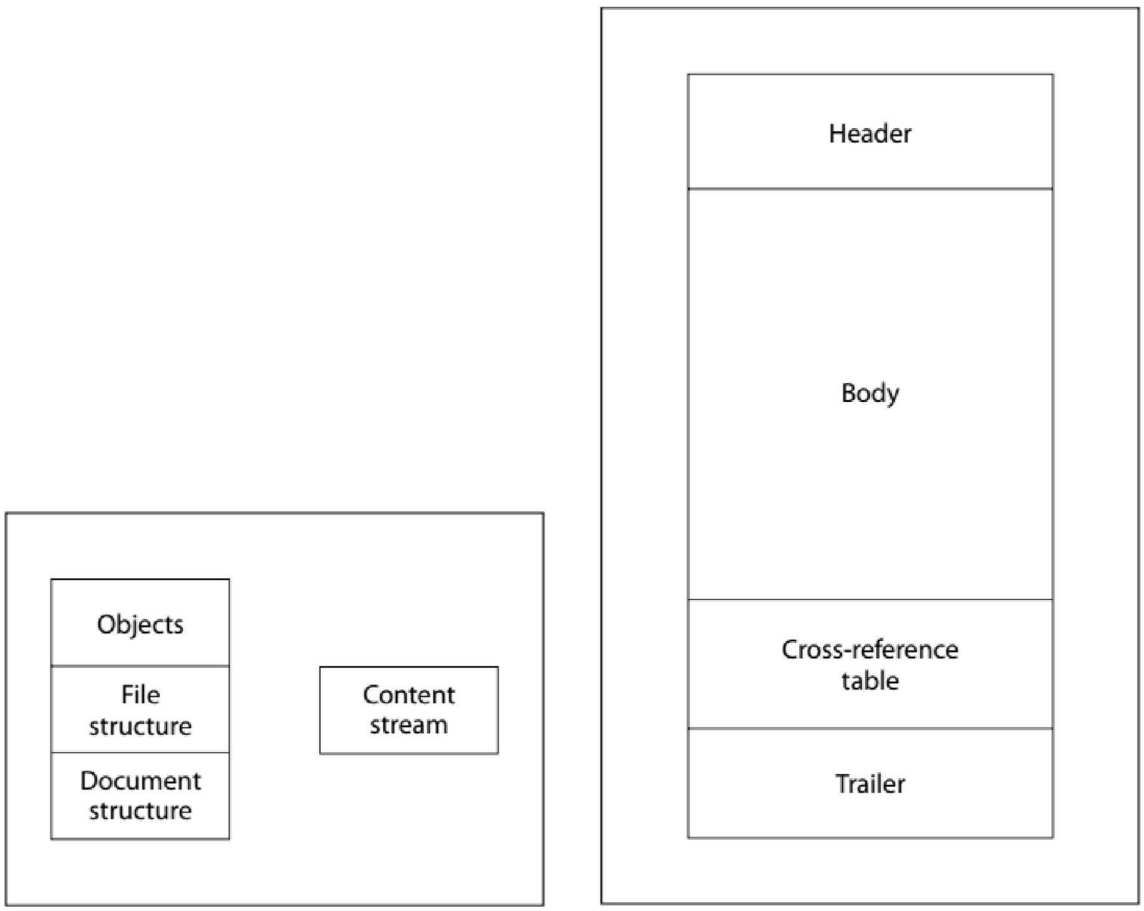

Fig. 2. Components (left) and structure (right) of a PDF file, accessed from (Incorporated, 2006).

Please cite this article in press as: Sloan, T., Hernandez-Castro, J., Dismantling OpenPuff PDF steganography, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.03.003

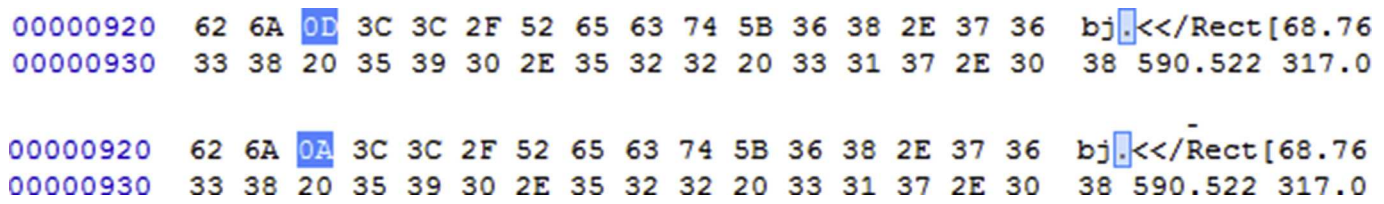

Fig. 3. Hex analysis of EOL components modified by OpenPuff.

are used to denote newlines in the metadata of a file. These are represented by 0×0A and 0x0D in hex, which we can use as a key identifier for the detection of OpenPuff PDF steganography. This is determined by our testing of OpenPuff's embedding impact over clean PDF files.

As shown in Fig. 3, the CR and LF characters are flipped once modified. Embedding modifications by OpenPuff will not be detected through analysis of file metadata via PDF parsers. Use of PDF-parser alongside WinMerge revealed identical files. This emphasises that the OpenPuff embedding algorithm requires more focused analysis.

When the original file employs exclusively the 0×0A character, OpenPuff operates in exactly the opposite fashion. Again, it is the coexistence of both 0x0D and 0×0A markers that reveal hidden contents, because the tool never flips all available values, as it has a maximum capacity of 50%. Because of this, a method can be constructed to perform analysis of PDF files and extract CR + LF characters. We have developed a proof of concept for this which is shown alongside our testing in Section Testing.

In addition to our detection scheme, we constructed an estimator to determine the maximum possible number of bytes that could be embedded within each file. The following calculation estimates the maximum possible embedded size by using least squares (Lu et al., 2004).[1]

$$maxhiddensize = 0.2204 \times occurf11 + 0.2117 \times occurf21 + 0.2115 \times occurf31 - 194.6563$$[2]

Table 2 shows a sample of 10 files with the given maximum capacity for each PDF file and our scheme's estimation. Each file is from a self-created dataset using our proof-of-concept (Appendix A).

Testing

To examine the accuracy of the distinguisher, two separate rounds of testing are performed. The first round of testing uses a small dataset and non-random data embedding. The second series of tests comprises a significantly larger dataset and random embedded data.

First round of testing

In the first round of testing, we obtained a total of 3000 PDF files. These were accessed through a webcrawler that traversed three separate locations. The first was a random selection of urls generated from the DMOZ archive (AOL, 2015). The second dataset of PDF files was downloaded from the Archive.org website and the third set of PDFs were obtained from Google Scholar. These were tested in three batches of size 1,000 candidate PDFs which were run through the OpenPuff embedding algorithm and successful carriers were embedded with a small txt file that contained a single letter 'a'. This resulted in a total of 50 successful candidate carrier files that had information embedded via OpenPuff. In addition to this, false positive rates for the distinguisher were also tested. Tables 3 and 4 provide an overview of these tests.

The selection of a candidate PDF file for OpenPuff appears to be a highly selective process, based on the small number of successfully modified files (approximately 1.6%). Due to the exclusive modification of white-space characters, a PDF file can only be used if the file contains enough carrier bytes.

Second round of testing

The second round of testing included a significantly larger dataset. These were downloaded in three separate instances from Archive.org, using the search phrases "PDF", "Computing", and "Medical". The new datasets comprised a total of 13,000 PDF files. From this, 135 were eligible candidates for OpenPuff steganography.

For each eligible PDF file, OpenPuff embedded random data at the maximum capacity offered by the tool. The distinguisher demonstrated results similar to that shown for the first series of tests with again 100% accuracy for each stego-object analysed (see Table 5).

False positives produced similar results to those shown on the first series of tests. A false positive rate of 0.5% will ensure that investigative time spent addressing these flagged objects is minimal (see Table 6).

This method of false positive testing covers a diverse range of PDF files. These are from many different creators and sources, and in some cases do not specifically follow official standards for the generation of PDF files.

Table 2
Estimator results.

File	Maximum Capacity (bytes)	Estimation (bytes)	Error (%)
Test File 1	80	90.64	+13.3
Test File 2	112	110.74	−1.1
Test File 3	64	54.25	−15.2
Test File 4	544	545.19	+0.2
Test File 5	224	218.34	−2.5
Test File 6	864	858.92	−0.5
Test File 7	80	84.01	+5.0
Test File 8	1232	1219.72	−0.9
Test File 9	80	76.62	−4.2
Test File 10	416	412.68	−0.7

Table 3
Accuracy results for first series of tests.

Datasets	Number of files	Processed by OpenPuff (%)	Detection Rate
Dataset 1	1000	10 (1.0%)	100%
Dataset 2	1000	13 (1.3%)	100%
Dataset 3	1000	27 (2.7%)	100%

Table 4
False positive results for first series of tests.

Number of files	False Positives/Negatives	Accuracy
3000 (FP test)	13/1	99.6%

[1] The full script can be shown in Appendix A.
[2] The full script can be shown in Appendix A.

Please cite this article in press as: Sloan, T., Hernandez-Castro, J., Dismantling OpenPuff PDF steganography, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.03.003

Table 5
Accuracy results for second series of tests.

Datasets	Number of files	Processed by OpenPuff (%)	Detection Rate
Dataset 1	5000	56 (1.1%)	100%
Dataset 2	5000	67 (1.3%)	100%
Dataset 3	3000	12 (0.4%)	100%

Table 6
False positive results for second series of tests.

Number of files	False Positives/Negatives	Accuracy
13000 (FP test)	68/0	99.5%

Impact of findings

A total of 16,000 PDF files were downloaded and examined using our proposed distinguisher. The results of our tests alongside other existing attacks in literature reveal an unprecedented vulnerability in a tool that is surprisingly highly recommended. This is the second identified case in which OpenPuff embeds hidden data through manipulation of metadata components. This shows that it may work in a similar fashion for other embedding algorithms for OpenPuff. If other formats can also be attacked using similar techniques, then any use of the tool would be inadvisable. There appears to be a significant lack of PDF steganography tools, not only in academic literature, but also generally available to the public. This may result in people hiding sensitive information with OpenPuff, which as we have shown can be detected. As a result, it is important to stress that this tool should not be used in any security setting.

Conclusion

In this paper, we present the first attack against the PDF component of the OpenPuff embedding algorithm. The high accuracy of our distinguisher paired with the simplicity of the attack raises serious concerns about the tool's security and its true steganographic capabilities. In light of our results, we believe that the PDF component of OpenPuff should be considered completely broken and therefore not fit for purpose. Having studied the way in which OpenPuff hides data over PDF files, many similarities appear with previous results over MP4. This leads us to believe that other methods of OpenPuff steganography will be performed via only the most simplistic metadata manipulations. As a consequence of this and until further research is carried out, we strongly discourage the use of the OpenPuff tool and dispute the author's claim that this tool is “suitable for highly sensitive data covert transmission”.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme, under grant agreement No.700326 (RAMSES project). The authors also want to thank EPSRC for project EP/N024192/1, which partly supported this work.

Appendix A. Python Implementation of OpenPuff PDF Classifier

```
import re
import sys
import os
import glob

for filename in glob.glob('*.pdf'):

    with open(filename, "rb") as f:


        stream=f.read()
        f10 = re.findall( b'\x65\x6E\x64\x73\x74\x72\x65\x61\x6D\x0D', stream) #Counting "endstream+CR"
        occurf10=len(f10)
        f11 = re.findall( b'\x65\x6E\x64\x73\x74\x72\x65\x61\x6D\x0A', stream) #Counting "endstream+LF"
        occurf11=len(f11)
        f20 = re.findall( b'\x65\x6e\x64\x6F\x62\x6A\x0D', stream) #Counting "endobj+CR"
        occurf20=len(f20)
        f21 = re.findall( b'\x65\x6e\x64\x6F\x62\x6A\x0A', stream) #Counting "endobj+LF"
        occurf21=len(f21)
        f30 = re.findall( b'\x20\x6F\x62\x6A\x0D', stream) #Counting " obj+CR"
        occurf30=len(f30)
        f31 = re.findall( b'\x20\x6F\x62\x6A\x0A', stream) #Counting " obj+LF"
        occurf31=len(f31)

    hidden_contents=((occurf10*occurf11)+(occurf20*occurf21)+(occurf30*occurf31))>0

    if hidden_contents:
        print "This PDF file has contents hidden with OpenPuff"
        maxhiddensize=0.2204*occurf11+0.2117*occurf21+0.2115*occurf31-194.6563
        print "Our estimate of the maximum size of the embedded data is", maxhiddensize, "bytes"
    else:
        print "This PDF file does not have contents hidden with OpenPuff"
        if ((occurf10*occurf20)==0):
            print "It looks like a scanned or a compressed file"
```

References

Alizadeh-Fahimeh, F., Canceill-Nicolas, N., Dabkiewicz-Sebastian, S., Vandevenne-Diederik, D., et al., 2012. Using Steganography to Hide Messages inside PDF Files.

AOL, 2015. Dmoz Open-content Directory. https://www.dmoz.org//. Accessed 11 April 2018.

Fridrich, J., Goljan, M., 2002. Practical steganalysis of digital images: state of the art. In: Electronic Imaging 2002, International Society for Optics and Photonics, pp. 1–13.

Gallagher, S., 2012. Steganography: How Al-qaeda Hid Secret Documents in a Porn Video. www.arstechnica.com/business/2012/05/steganography-how-al-qaeda-hid-secret-documents-in-a-porn-video/ (Accessed 19 April 2017).

Hosmer, C., 2012. Data Hiding and Steganography Report 2012. http://embeddedsw.net/doc/OpenpuffpaperDathidingandsteganographyannualreport2012.pdf (Accessed 11 April 2018).

Incorporated, A.S., 2006. Pdf Reference, Version 1.7, sixth ed. Online. http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pds_reference_1–7.pdf.

King, J.C., 2005. Adobe Introduction to the Insides of Pdf.

Lingjun, L., Liusheng, H., Wei, Y., Xinxin, Z., Zhenshan, Y., Zhili, C., 2008. Detection of word shift steganography in pdf document. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks, ACM, p. 15.

Liu, B., Xu, E., Wang, J., Wei, Z., Xu, L., Zhao, B., Su, J., 2011. Thwarting audio steganography attacks in cloud storage systems. In: Cloud and Service Computing (CSC), 2011 International Conference on, IEEE, pp. 259–265.

Lu, P., Luo, X., Tang, Q., Shen, L., 2004. An improved sample pairs method for detection of lsb embedding. In: International Workshop on Information Hiding. Springer, pp. 116–127.

Oliboni C., Openpuff v4.00 steganography and watermarking, EmbeddedSW, 18/04/2017.

P. Paganini, Hackers Used Data Exfiltration Based on Video Steganography. http://securityaffairs.co/wordpress/30624/cyber-crime/hackers-used-data-exfiltration-based-video-steganography.html/. (Accessed 18 April 2017).

Project, T.G., 2017. The Guardian Project. https://guardianproject.info/ (Accessed 19 April 2017).

Provos, N., Honeyman, P., 2003. Hide and seek: an introduction to steganography. Secur. Priv. IEEE 1 (3), 32–44.

Rafat, K.F., Sher, M., 2013. Secure digital steganography for ascii text documents. Arab. J. Sci. Eng. 38 (8), 2079–2094.

Shachtman, N., 2010. Fbi: Spies Hid Secret Messages on Public Websites. http://www.wired.com/2010/06/alleged-spies-hid-secret-messages-on-public-websites/.

Sloan, T., Hernandez-Castro, J., 2015. Steganalysis of openpuff through atomic concatenation of mp4 flags. Digit. Invest. 13, 15–21.

Young, L., 2015. The Dark Side of Steganography. http://spectrum.ieee.org/tech-talk/telecom/security/the-dark-side-of-steganography/.

Zhong, S., Cheng, X., Chen, T., 2007. Data hiding in a kind of pdf texts for secret communication. IJ Netw. Secur. 4 (1), 17–26.

Zukerman, E., 2013. Review: Openpuff Steganography Tool Hides Confidential Data in Plain Sight 2013. http://www.pcworld.com/product/1252470/openpuff.html (Accessed 19 April 2017).

Please cite this article in press as: Sloan, T., Hernandez-Castro, J., Dismantling OpenPuff PDF steganography, Digital Investigation (2018), https://doi.org/10.1016/j.diin.2018.03.003