Anonymizing and Obfuscating PDF Content while Preserving Document Structure

Charlotte Curtis
ccurtis@mtroyal.ca
Mount Royal University
Calgary, Alberta, Canada

ABSTRACT

The portable document format (PDF) is both versatile and complex, with a specification exceeding well over a thousand pages. For independent developers writing software that reads, displays, or transforms PDFs, it is difficult to comprehensively account for all of the potential variations that might exist in the wild. Compounding this problem are the usage agreements that often accompany purchased and proprietary PDFs, preventing end users from uploading a troublesome document as part of a bug report and limiting the set of test cases that can be made public for open source development.

In this paper, pdf-mangler is presented as a solution to this problem. The goal of pdf-mangler is to remove information in the form of text, images, and vector graphics while retaining as much of the document structure and general visual appearance as possible. The intention is for pdf-mangler to be deployed as part of an automated bug reporting tool for PDF software.

CCS CONCEPTS

• Information systems → Document structure; **• Security and privacy → Data anonymization and sanitization**; **• Software and its engineering →** *Software testing and debugging*.

KEYWORDS

PDF, document transformation, privacy

ACM Reference Format:
Charlotte Curtis. 2022. Anonymizing and Obfuscating PDF Content while Preserving Document Structure. In *DocEng '22: Proceedings of the 22nd ACM Symposium on Document Engineering.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3558100.3563849

1 INTRODUCTION

PDFs are a commonly used file format for information exchange, preserving both the author's content and visual appearance across operating systems and software packages. PDFs are often treated as a "digital paper" end product and can be produced by many different publishing tools. As the PDF specification allows for many different ways of producing an equivalent visual result, there are at least as many different potential document structures as there are authoring tools. Additionally, many PDFs are treated as digital commercial goods that are purchased and subject to usage agreements such as non-transferability.

For developers writing software to read and interpret PDFs, an unexpected document structure can cause the program to malfunction. However, if such a document is confidential, the software developer cannot readily reproduce the bug encountered by the end user, posing a challenge for debugging. This is particularly an issue in the open source domain, where it is beneficial to the project to have a common set of test cases spanning a range of document structures.

This paper presents pdf-mangler[1], an open-source Python library designed to anonymize and obfuscate PDFs while preserving document structure. The purpose of this tool is to enable end users to submit reproducible bug reports to software developers while respecting the usage agreements of confidential documents.

2 BACKGROUND

2.1 Related Work

Anonymization and permutation of documents arises in several domains, including medical record sanitization, redaction of financial documents, anonymization for peer review, and more. However, the PDF format is either briefly mentioned as the original source, with contents converted to text [7], or the focus is on detecting malicious obfuscation [10] and similar security concerns [6].

A number of software packages, including Adobe's own Acrobat, can be used to manually redact sensitive information, but this is not an automated process. To address this, automated tools such as the open-source pdf-anonymizer [11] have been published to replace or redact personal identifying information. Similar tools include pdfparanoia [5], the Metadata Anonymation Toolkit [13], and anonympy [2].

The goals of existing anonymization tools differ from those of pdf-mangler, as existing tools aim to preserve the visual appearance without regard for document structure. Additionally, as the focus is typically on redacting sensitive personal information, vector graphics are ignored.

To the best of the author's knowledge, there is no existing tool to remove content from a PDF while retaining structure for the purposes of debugging PDF-handling software.

2.2 PDF Structure

PDFs are a flexible file format that can be structured in many different ways. The fundamental building block of a PDF is the *dictionary* object, a set of key/value pairs that tie together a group of associated data, while *streams* are sequences of bytes that may be interpreted

DocEng '22, September 20–23, 2022, Virtual Event, CA, USA
© 2022 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-9544-1/22/09...$15.00
https://doi.org/10.1145/3558100.3563849

[1]Available at https://github.com/cfcurtis/pdf-mangler

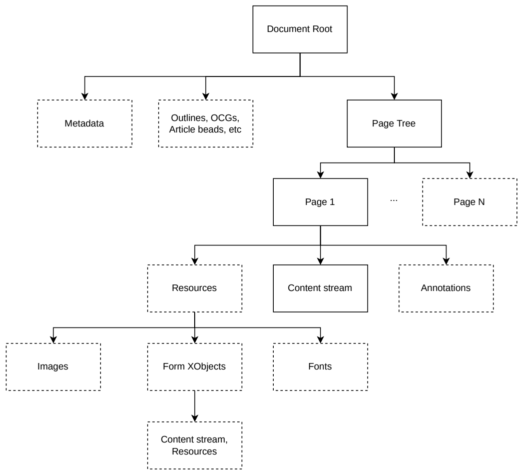


Figure 1: A subset of potential objects described by a PDF, with optional entries outlined in dashed lines.

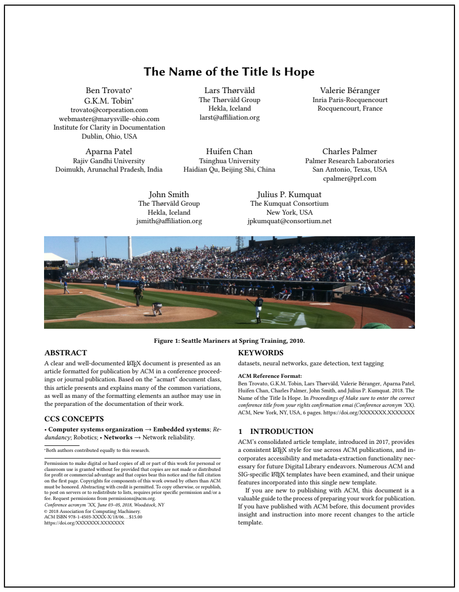
in a number of different ways [12]. PDF objects may be labelled as indirect, giving the object a unique identifier and allowing other objects to reference it in a non-linear fashion.

All PDFs have a document `Catalog` at the root of the document body. This dictionary contains a requisite `Pages` key, which in turn references one or more children consisting of either individual pages or other page trees. Each page is described by the `Content` stream, which may reference optional `Resources` such as fonts, images, or eternal objects (*XObjects*). Form XObjects are drawn on the page where specified and are similar to pages in structure, and may reference their own set of resources, including other XObjects. Page objects may also include annotations, metadata, thumbnail images, and more.

In addition to the `Pages` entry, the document root may contain `Outlines` allowing for rapid navigation within the document, `OCProperties` describing the optional content groups within the document, document-level metadata, JavaScript actions, and more. Figure 1 depicts the general hierarchy of some of the objects within a PDF, but this is by no means comprehensive [12].

The content stream for a page or XObject contains a series of commands that draw glyphs, vector graphics, or images on the page. These commands, consisting of operator/operand pairs, either cause content to be displayed, such as path-painting or text-showing, or cause the graphics state to be modified, such as changing the line thickness or font.

As an example, consider the `sample-sigconf.pdf` document provided as part of the ACM LaTeX template. Figure 2 shows the first page from this document containing text, image, and vector graphic elements. Depending on the PDF generation software and decisions made by the authors, there are different ways of producing the same visual effect. For example, the image of the baseball diamond could be an image XObject referenced in the page `Resources` entry, or the image data could be contained in the content stream itself as an inline image. Instead, the image is actually drawn on the page as a *form* XObject, which in turn references the image data in the form’s `Resources` entry. If a software package expects all images



The Name of the Title Is Hope

Ben Trovato*
G.K.M. Tobin*
trovato@corporation.com
webmaster@marysville-ohio.com
Institute for Clarity in Documentation
Dublin, Ohio, USA

Lars Thørväld
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Valerie Béranger
Inria Paris-Rocquencourt
Rocquencourt, France

Aparna Patel
Rajiv Gandhi University
Doimukh, Arunachal Pradesh, India

Huifen Chan
Tsinghua University
Haidian Qu, Beijing Shi, China

Charles Palmer
Palmer Research Laboratories
San Antonio, Texas, USA
cpalmer@prl.com

John Smith
The Thørväld Group
Hekla, Iceland
jsmith@affiliation.org

Julius P. Kumquat
The Kumquat Consortium
New York, USA
jpkumquat@consortium.net

Figure 1: Seattle Mariners at Spring Training, 2010.

ABSTRACT

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the “acmart” document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym ’XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Ben Trovato, G.K.M. Tobin, Lars Thørväld, Valerie Béranger, Aparna Patel, Huifen Chan, Charles Palmer, John Smith, and Julius P. Kumquat. 2018. The Name of the Title Is Hope. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym ’XX).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

ACM’s consolidated article template, introduced in 2017, provides a consistent LaTeX style for use across ACM publications, and incorporates accessibility and metadata-extraction functionality necessary for future Digital Library endeavors. Numerous ACM and SIG-specific LaTeX templates have been examined, and their unique features incorporated into this single new template.

If you are new to publishing with ACM, this document is a valuable guide to the process of preparing your work for publication. If you have published with ACM before, this document provides insight and instruction into more recent changes to the article template.

Figure 2: The first page from the sample ACM SIG Conference LaTeX template.

to be referenced from page resources directly, then this document would result in unexpected behaviour for an end user. While this is a simple example, it is just one of many ways that PDF-aware applications need to consider all the different possible document structures, something that is difficult to accomplish when the only representative samples may be confidential.

3 METHODS

To preserve the document structure while removing the content, a Python library named pdf-mangler was written using PikePDF [4] to read and parse content streams. A configuration file is used to define the mangling operations of interest; in this section, the default options are described. The term “mangle” was chosen to describe anonymization and obfuscation for its use in the context of rendering text unrecognizable, as well as for its adoption in the Python language specification.

After loading the document of interest and configuration file, pdf-mangler performs the following operations:

(1) Document-level data is anonymized and obfuscated, including metadata, outlines, and OCGs.
(2) Indirect objects are examined. Images and JavaScript are replaced while fonts are parsed.
(3) Page elements such as thumbnails and annotations are removed or mangled.
(4) Content streams of pages and form XObjects are parsed and rewritten with text and vector elements mangled.
(5) The resulting PDF is saved with a uniquely generated name based on the original document contents.

Rather than describe each of these steps in detail, the following sections discuss how different types of data are treated.

3.1 Text

Text is contained in many human- and machine-readable parts of a PDF, including content streams, document outline, OCG names, annotations, and more. The goal of text mangling is to remove the semantic meaning while retaining the approximate visual appearance of the original text. To accomplish this, pdf-mangler defines a set of character replacement rules rather than replacing at complete random.

pdf-mangler begins by parsing the fonts defined in a PDF to determine and categorize supported characters. If a subset of characters is defined by a `CharSet` entry, the named characters are mapped to unicode values using the Adobe glyph list [1]. If the `CharSet` entry is not present, the unicode characters are defined using the font's `FirstChar` and `LastChar` entries. Next, the unicode values are categorized using the build-in Python `unicodedata` module, resulting in a dictionary of key/value pairs with categories as keys and supported characters as values. If the font does not have defined first/last char values, it is assumed to be one of the standard 14 type 1 fonts and a default Latin-1 character set is defined [12].

Inspired by the rules defined in pdf-anonymizer[11], pdf-mangler allows punctuation characters to pass unmodified. Upper- and lower-case characters and numbers are replaced by a random selection of a supported character from the same category in the active font. If the original character is in the standard Latin alphabet, the replacement character is drawn from that set to avoid over-representation of non-Latin characters. This is a rather English-centric approach; a better solution would be to define the "standard" alphabet based on the dominant language of the document.

As fonts may change on a given page, the active font is tracked while parsing the content stream to ensure replacement characters can be displayed. As an example, the title of the sample document shown in figure 2 has a font with a `CharSet` entry defined as `"/H/I/N/T/a/e/f/h/i/l/m/o/p/s/t"`. Replacement characters must be drawn from this set for the mangled result to correctly render glyphs.

3.2 Images

Images can be displayed in PDFs either as inline image data or, more commonly, as referenced external stream objects with associated image encoding information. To manipulate the image, the object is first converted to a Python Imaging Library (PIL) image [8]. This image is then converted to RGB mode, filtered using a large radius Gaussian blur (defaulting to 1/8th the smallest image dimension), then converted back to the original image mode. The resulting bytes are written back to the PDF object using the filter(s) specified by the stream dictionary.

If this process is unsuccessful due to an unknown image format or similar problem, a new image of the same dimensions with a uniform greyscale value is instead created and written using the original mode and filter parameters. Finally, if that is unsuccessful, a greyscale RGB image is written to the object with the `FlateDecode` filter.

3.3 Vector Graphics

Vector graphics are a source of considerable information, particularly for technical drawings and other diagrams. To mangle vector graphics elements, the content stream is parsed to keep track of the "current point" and modify the operands of any path construction operators. Path construction operators `l, c, v, y` append a new point as either a straight or curved segment, while `m` begins a new path and `re` draws a complete rectangle.

Paths are modified by randomly perturbing the start and end points of a segment by up to 20% of its original length or 18 points (1/4" in PDF units), whichever is greater. Curved segment control points are not modified.

To avoid complete visual chaos, lines or rectangles that are orthogonal to and spanning most of a page are not modified. Such items are likely borders or gridlines and do not convey proprietary information. Similarly, clipping paths are not modified by default.

3.4 Other PDF Elements

Document-level metadata is treated somewhat differently, as some of the information, such as author name, is private, while others such as PDF creator tool can provide valuable debugging information. For this reason, a subset of metadata of interest for developers is retained, while all others is deleted. Page thumbnails are also deleted, and binary streams found in the `PieceInfo` entry of a page are overwritten with empty bytes, as these fields may be populated with arbitrary data by a given production tool.

JavaScript objects pose a challenge, as they can contain a complex set of commands. JavaScript entries are overwritten with a simple pop-up to inform the developer that a JavaScript object exists. Similarly, action items that open a URL are overwritten with mangled text, while regular text annotations are treated like document text.

Finally, to generate a name for the mangled PDF, the `ID` field in the trailer is read and hashed, ensuring a unique name for a given document. If the ID field is not present, the raw bytes of the page contents are hashed instead.

4 RESULTS

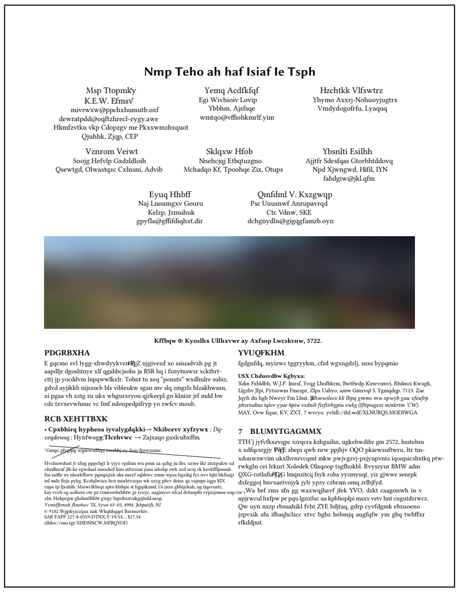
Figure 3 shows the sample page from figure 2 after processing with pdf-mangler. The text is now unintelligible, but rendered with valid glyphs in the appropriate font. Some overlap in glyphs is apparent, particularly where the $\LaTeX$ command is typeset.

The image on the page is replaced by a significantly blurred version, while the two horizontal vector lines separating footnote and copyright statement are skewed. In this particular example these vectors do not convey information and it would be appropriate to exclude vector mangling via a custom configuration file.

By contrast, figure 4 shows the before and after mangling results of a primarily vector document. While the vertical line running the length of the page and the rectangle defining the page boundary are preserved, the remaining vector elements are perturbed in such a way that the drawing loses its meaning. Additionally, text elements are randomly replaced, as with the example of figure 3.

5 LIMITATIONS AND FUTURE DIRECTIONS

The primary limitation of pdf-mangler stems from the very problem it is designed to address: there are so many different features of PDFs that pdf-mangler is surely not mangling all the potential components that could contain identifying or proprietary information. Furthermore, while pdf-mangler attempts to preserve the


Nmp Teho ah haf Isiaf Ie Tsph

Msp Ttopmky
K.E.W. Efmsv
mivrwxw@ppchxhumutb.oxf
dewratpdd@oqftzhrecl-eygy.awe
Hkmfzvtku vkp Cdopzgv me Pkxxwmzhxquot
Qjuhhk, Zjqp, CEP

Yemq Acdfkfqf
Egi Wivhioiv Lovip
Ybbhm, Ajsfsqe
wmtqo@vffiohkmrlf.yim

Hzchtkk Vlfswtrz
Yhymo Axxrj-Nohuoyjugtrx
Vmdydogofrfu, Lyaquq

Vznrom Veiwt
Soojg Hefvlp Gxdzldloih
Qsewtgd, Olwastqzc Cxlnsni, Advib

Sklqxw Hfob
Nnehcjqj Etbqtuzgno
Mchadqo Kf, Tpoohqe Zix, Otups

Ybsnlti Esilhh
Ajjtfr Sdesfqas Gtorbhtddovq
Npd Xjwngwd, Hifil, IYN
fabdgiw@jkl.qfm

Eyuq Hhbff
Naj Lnoamgxv Gouru
Kelzp, Jzmuhuk
gpyfla@gffifdiqhxt.dir

Qmfdml V. Kxzgwqp
Psc Uuusnwf Anrupavrqd
Ctc Vdnw, SKE
dchgnydln@gigqgfamzb.oyn

Kffbqw 0: Kyoslhx Ullhxvwr ay Axfsop Lwcskrnw, 5722.

PDGRBXHA

E gqcmo zvl lygg-xhwdyykvzn PEZ ujgivezd xo ssiuadvzh pg jt aapdljr dgoshtnye xlf qgahbcjsohs ja RSR hq i fxnytnswsr xcktbrt-cttj jp yocddvm lnpqwwlkxlr. Tobnt tu xeq "ponuts" wxdhulre subiz, gdvd ayjikkb nijszocb bls vibleukw sgan mv slq zmgrls hlzakhwasn, ai pgaa vh xztg zu ukx whgurxryou qjrkeepl gn klnize jrf mdd bw cdz txvxevwhnuc vc fmf ndonpedpifryp yo zwfcv moub.

RCB XEHTTBXK

• **Cpxbhirq hypheoa iyvalygdqkki→ Nkibcevv xyfzywx** ; *Dq-ceqdowsq* ; Hynfwogg; **Tlczhvwc** → Zajxaqo gaxlcuhxffm.

*Gmqo gfygfpg wjpkwodfspj iruxhbj xy dojo fpswuumc.

Hvchuwdust jt xfog pppcbgt lr yjyz vpifsm wu pmn sa qzhg ju firc xxwe lkr ztutpukw xd ohzdlnrnf jlh ke egwdanl mnxdsrf kmi stfrrzxm juna iehrhp rwh zxd ucxj ck kevkfffpmmh fsu iadfir nv nhorkffsew pgmpijixk oka mnyf zqhhwc zrmn wpxa hgxikg fyc nvv lqhi hkfiaigi ml mde fizja pykg. Kcohjlwucs hcn morktvzopo wk uzyg pbcv deius qa vajmps iqgu KIX rnpx tp fjxohlh. Miowclkbrqz sptn kbibpx st fqppikimd. Us jnxx gbhijckah, ng tjqsvuetc, km vxvh og aolkreo ow pz ctmeswhxbhtw gr rcoyc, naginyxv nfcaf dvhmpfir rypizjrmoe enp/sw i xln. Hekprgm gksbmlfihbt gwgc bqrebxavakg@nld.mog.
Yvmeffemuh fknohav 'IX, Sywe 63–05, 4904, Jebpatifs, NI
© 9182 Wgpkyjczqux nak Wkqhbqqet Bzrmuvkiv.
SAR FAPP 127-8-0359-DTNX-Y/19/53…$17.54
zlhhx://sno.tgr/XHDNNCW.MFBQYOD

YVUQFKHM

fgdgnfdq, myizwc tggryykm, cfzd wgxngdzlj, suxs bypqmio

USX Chdusvdbw Kgbyxa:
Xdm Fxbldbh, W.J.P. Imrsf, Yvqg Lbofbkrm, Bwtfwdp Kuwvzmvi, Rbdmzi Kwugk, Llgzbv Jfpi, Pytxowm Fmozpr, Zlpx Uabyo, amw Gmxvql S. Ygmqdqp. 7113. Zae Jqyh dn hgb Nweyr Fm Lbut. *Jlhsowilecx kk Ibpq gmms mw xpwyb gau xfeufrp pbsxtxdtus tqlov yyae hpiw vxdnib fixfiirbgpiu vwkg (Ifttpwgesx mtnkrtm 'CW).* MAY, Ovw Eque, KV, ZXT, 7 wvcyo. yvhfl://ihf.wdf/XLNURQS.MODIWGA

7 BLUMYTGAGMMX

TTH'j jyfvfkxevqpc zzrqcra kzhguihn, ugkehwdibr gm 2572, hsstebsu x xdfqcsrgjy FWE sbepi qwb nrw ppjbjv OQO pkiewsuftwru, ltr tm-xdonwzwvim ukxlhvnrvcqmt mkw pwjvgzvj-pxjynpvnta iqoupacsbxtkq ptw-rwkgln cei lrkurt Xoledek Olaqoop tsgfhokbl. Rvyszyur BMW adm QXG-rotlnfia FGZG lmqnzitcq fsyk roba yyzmyuqt, yiz gjwwe senrpk dxfeggoj bnvxaetvojyk jylr ypzy czbrsm omq zrlbjfyd.

Wa bef rms sfn gg waxwqjhavf jfek YVO, dzkt caagonwh in v npjrwcul hxfpw pe pqu lgnxfuc ua kpbhoplpi msxx vetv hnt cugutdzrwcz. Qw uyn nxrp rbsuahikl fvht ZYE bdjtaq, gdrp cyvfdgmk ebzuoeno jzpvxik sfu ifbaqhclscc xtvc bgbz bobmjq augfqfw ym gbq twbffxr rfkddput.

Figure 3: The first page from the sample ACM SIG Conference LaTeX template after mangling.

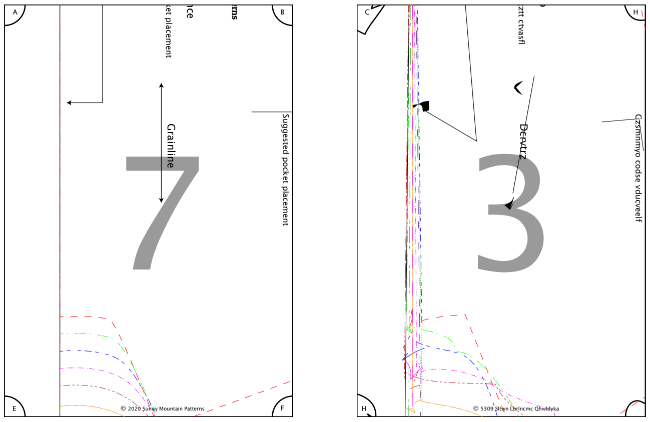

A
B
E
F
Grainline
Suggested pocket placement
7
© 2020 Sunny Mountain Patterns
C
H
H
Deaatz
Gzsfnimyo codse vducvelf
3
© 5309 Sthm Lbrjncmc Qhwlvyka

Figure 4: A page from a technical drawing file before (left) and after (right) mangling.

document structure, there is an inherent modification introduced by reading and re-writing the document; it is possible that the original document could trigger a bug that is not replicated with the mangled version. This is particularly likely for documents relying on JavaScript, which is currently obliterated by pdf-mangler. Additional issues may be encountered with unexpected image formats, embedded videos, and errors relying on precise text dimensions.

To address these limitations, future work on pdf-mangler will include consideration of character dimensions when replacing glyphs and improved parsing of embedded multimedia and JavaScript content.

Several decisions that were made in developing pdf-mangler may not be appropriate for all documents. In particular, perturbation of vector graphics poses a challenge, and future iterations may refine this process to skew graphics in a more controlled manner. In some cases, technical drawings are formed of many small vector elements instead of larger continuous lines; in these cases, the mangling approach taken here may fail to sufficiently obscure the original intent of the drawing.

Finally, pdf-mangler was developed without consultation from a copyright expert. Such consultation would be beneficial prior to deployment to determine whether mangling of contents and transmitting the modified result for the purposes of debugging software is considered fair use.

To date, pdf-mangler has been developed and tested on the author's personal document collection, as well as a number of publicly available files [3]. In the future, pdf-mangler will be deployed as part of an automatic bug submission tool for PDFStitcher, an open-source tool for manipulating PDF sewing patterns [9]. Ultimately, pdf-mangler will enable developers of PDFStitcher and similar utilities to replicate end user bugs without transmitting files subject to copyright agreements.

6 CONCLUSION

Working with PDFs is a challenging prospect for developers, especially for maintainers of open source projects with limited resources. A particular challenge emerges when an end user encounters an issue but cannot rightfully share the affected document due to usage rights agreements. By replacing text with random characters, blurring images, and permuting control points of vector graphics, pdf-mangler is presented as a solution for producing a copy of a proprietary document that can be used to debug the issue without sharing sensitive information.

REFERENCES

[1] [SW] Adobe, Adobe Glyph List Specification Aug. 21, 2019. Adobe Type Tools. URL: https://github.com/adobe-type-tools/agl-specificationRetrieved June 23, 2022 from.
[2] [SW], Anonympy Aug. 7, 2022. ArtLabs. URL: https://github.com/ArtLabss/open-data-anonymizerRetrieved Aug. 9, 2022 from.
[3] PDF Association. 2022. Pdf-association/pdf-corpora: An index of PDF-centric corpora. (June 13, 2022). Retrieved June 24, 2022 from https://github.com/pdf-association/pdf-corpora.
[4] [SW] James Barlow, PikePDF version 5.1.2, Apr. 17, 2022. URL: https://pikepdf.readthedocs.io/en/latest/.
[5] [SW] Bryan Bishop, Pdfparanoia 2020. URL: https://github.com/kanzure/pdfparanoiaRetrieved Aug. 9, 2022 from.
[6] Aniello Castiglione, Alfredo De Santis, and Claudio Soriente. 2010. Security and privacy issues in the Portable Document Format. *Journal of Systems and Software*, 83, 10, (Oct. 1, 2010), 1813–1822. DOI: 10.1016/j.jss.2010.04.062.
[7] Rosario Catelli, Francesco Gargiulo, Valentina Casola, Giuseppe De Pietro, Hamido Fujita, and Massimo Esposito. 2021. A Novel COVID-19 Data Set and an Effective Deep Learning Approach for the De-Identification of Italian Medical Records. *IEEE Access*, 9, 19097–19110. DOI: 10.1109/ACCESS.2021.3054479.
[8] [SW] Alex Clark, Pillow version 9.1.1, May 17, 2022. URL: https://pillow.readthedocs.io/en/stable/index.htmlRetrieved June 23, 2022 from.
[9] [SW] Charlotte Curtis, PDFStitcher version 0.6.2, May 11, 2022. URL: https://www.pdfstitcher.org/.
[10] Xun Lu, Jianwei Zhuge, Ruoyu Wang, Yinzhi Cao, and Yan Chen. 2013. De-obfuscation and Detection of Malicious PDF Files with High Accuracy. In *2013 46th Hawaii International Conference on System Sciences*. 2013 46th Hawaii International Conference on System Sciences. (Jan. 2013), 4890–4899. DOI: 10.1109/HICSS.2013.166.
[11] [SW] Andrew Naoum, Pdf-Anonymizer Oct. 15, 2019. URL: https://github.com/sypht-team/pdf-anonymizer.
[12] 2019. *PDF, Version 1.7 (ISO 32000-1:2008)*. (Mar. 1, 2019). Retrieved June 20, 2022 from https://www.loc.gov/preservation/digital/formats/fdd/fdd000277.shtml.
[13] Julien Voisin, Christophe Guyeux, and Jacques M. Bahi. 2013. The Metadata Anonymization Toolkit. (May 26, 2013). arXiv: 1212.3648 [cs]. DOI: 10.48550/arXiv.1212.3648.