Story Beyond the Eye: Glyph Positions Break PDF Text Redaction

MAXWELL BLAND, ANUSHYA IYER, and KIRILL LEVCHENKO, University of Illinois, Urbana-Champaign, USA

In this work we find that many current redactions of PDF text are insecure due to non-redacted character positioning information. In particular, subpixel-sized horizontal shifts in redacted and non-redacted characters can be recovered and used to effectively *deredact* first and last names. Unfortunately these findings affect redactions where the text underneath the black box is removed from the PDF.

We demonstrate these findings by performing a comprehensive vulnerability assessment of common PDF redaction types. We examine 11 popular PDF redaction tools, including Adobe Acrobat, and find that they leak information about redacted text. We also effectively deredact hundreds of real-world PDF redactions, including those found in OIG investigation reports and FOIA responses.

To correct the problem, we have released open source algorithms to fix trivial redactions and reduce the amount of information leaked by *nonexcising* redactions (where the text underneath the redaction is copy-pastable). We have also notified the developers of the studied redaction tools. We have notified the Office of Inspector General, the Free Law Project, PACER, Adobe, Microsoft, and the US Department of Justice. We are working with several of these groups to prevent our discoveries from being used for malicious purposes.

1 Introduction

Redaction is a centuries-old process of removing or obscuring parts of a document to prevent their disclosure. In the past this was performed by black marker or by physically cutting parts of the document out with scissors. However, with the move to digital, paperless representations of documents, the process of redaction is typically performed by software tools. The process of removing information from digital documents is error-prone, as demonstrated by several high-profile examples [22]. However, no existing work has considered the role the PDF file representation plays in redaction security.

In this work, we identify problems in PDF file text redaction. It is well known that the width of the characters in a proportional-width font can leak information about redacted text. However, we find PDF files include sub-pixel sized character position shifts, which can leak more information about redacted text than width alone. Failing to understand these shifts exist leads to incorrect or imprecise deredaction. We are the first to identify this problem.

This led us to measure the severity of information leaks in PDF text redactions. We discovered methods for breaking redactions occurring on documents processed by optical character recognition (OCR) software and find that *rasterizing* a document (converting the PDF to an image) increases the security of redactions but does not remove information leaks. We also find a typical PDF document authored in Microsoft Word leaks about 13 bits of information about a redacted surname. This is enough for attacker to consistently identify a single individual among 8,000 candidates (say, employees at a company or potential confidential informants in a gang), or ≈500 potential first initial, surname pairs from the 3.9 million possible given US census and Social Security Administration data.

We also consider *nonexcising* redactions which do not remove any text from the document. The text of these redactions can be copied and pasted from the source document. Our study and techniques located thousands of nonexcising redactions in US court documents. We notified both the US court system and the Free Law Project about these redactions.

Authors' address: Maxwell Bland, mb28@illinois.edu; Anushya Iyer, anushya2@illinois.edu; Kirill Levchenko, klevchen@illinois.edu, University of Illinois, Urbana-Champaign, Urbana, Illinois, USA.

We trace the source of the insecure redactions by examining 11 popular PDF redaction tools and find that two do not work at all. These two tools leave the text underneath the selection marked for redaction in the PDF document. The remaining tools, including those from Adobe Systems, remove the redacted text and replace it with a black rectangle. Without additional defenses, this practice leaks significant information about the redacted text.

We built a tool, Edact-Ray, which we use to identify, break, and fix redaction information leaks across millions of PDF files. During the deredaction stage, Edact-Ray allows an attacker to test which strings from a dictionary of candidates create the same PDF output as the original redacted PDF file. We applied Edact-Ray to study the impact of our findings on three publicly-available corpora of PDF documents.

We have notified organizations whose redacted documents we study and affected vendors of redaction tools and are working with several of these groups to remediate the problem. To address the discovered vulnerabilities, we release tools for identifying and securing poorly redacted PDF text at [anonymized for peer review].

The contributions of this work are:

- A survey of 11 popular PDF redaction tools. We find two tools have redaction features which do not actually remove text marked for redaction, making the redacted document vulnerable to copy-paste attacks. The remaining tools preserve glyph positioning information which can be used for glyph displacement attacks.
- The discovery of redacted information leaks due to sub-pixel sized glyph position shifts. We provide a precise analysis of the role and impact of these shifts on the process of deredaction.
- An information theoretic measurement of PDF redaction security. A redaction of a first name and surname from a voter registration database in a PDF produced by Microsoft Word's "Save as PDF" feature can leak up to 15.9 bits of information out of 19.6, meaning an adversary can correctly break such a redaction 38% of the time.
- Novel advisories regarding the security of PDF redactions and correct redaction methods. PDF redactions where the set of candidates may be reasonably constrained, e.g. a redaction of a politician's name, are not secure. Redactions of individuals' names where no other adjacent words are redacted are not secure.
- A survey of millions of publicly-available redacted documents. We find many of the documents provided by the US Courts, the Office of Inspector General (OIG), and Freedom of Information Act Requests (FOIA) are vulnerable. We found over 6,000 vulnerable nonexcising redactions. We found over 300 vulnerable excising redactions.
- An extensive notification of affected parties and the release of several software tools for the development of advanced redaction defenses. We have collaborated with the US government and third party software providers to correct and prevent the occurrence of discovered redaction vulnerabilities.

This rest of this paper is organized as follows. Section 2 gives background on how PDF text redactions leaks information. Section 3 surveys PDF redaction tools' preservation of redacted information. Section 4 describes Edact-Ray's implementation. Section 5 measures mutual information shared by redacted and non-redacted PDF text. Section 6 measures the number and diversity of redaction information leaks in public PDF documents. Section 7 discusses defenses and future work. Section 8 addresses related work and Section 9 concludes.

2 PDF Redaction Security

The security of PDF text redaction depends on the specification of the PDF document. We consider two types of PDF documents. One is a raster image of the original document. The other PDF

document type contains text data for both the font and the layout of each character (*glyph*) on the page.

We focus our discussion on non-raster PDF documents, however, the concepts presented apply to raster PDF documents. We consider the effect rasterization has on redaction information leaks when discussing it as a defense in Section 7.1 and include a discussion of different document scanners in Appendix H.1.

```
[...(Exh)-2(ibit A.)] TJ
          Positional Adjustment
```

Fig. 1. The TJ text showing operator, which specifies the glyphs to render and, by reference to a font object (not shown), their widths, along with any associated positional adjustments, given in text space units.

PDF documents can render text in numerous ways, including by use of a text showing operator, one of which (TJ) is depicted in Figure 1. The TJ operator takes as arguments a string of text and a vector of *positional adjustments* which displace the character with respect to a default position. This default position is usually a fixed offset from the previous character equivalent to the *advance width* of the previous character defined elsewhere in the PDF document.

For this paper, we converted the complex set of PDF text rendering operations into a uniform *intermediate representation*, consisting of a minimal set of metrics (e.g. font size) and a series of TJ operators.[1] The intermediate representation presents PDF text as, effectively, a set of advance widths and *glyph shifts* which are the sum of all the individual positioning operations applied to a glyph. This conversion was necessary to account the large number of ways in which text can be rendered in PDF documents.[2]

The positional adjustment in Figure 1 is -2 *text space units* between the *h* and *i* glyphs. Text space units express glyph shifts, where 1,000 units almost always[3] equals the point size of the font times 1/72 of an inch. For a 12-point font, 1 unit equals 1/6,000 of an inch (0.0042 mm).

Glyph advance widths and glyph shifts create a security concern. Section 3 finds that most PDF redaction tools replace text selected for redaction with a single large shift of the same width as the redacted text showing operator, creating the two significant security risks:

- The precise width of the redaction can be used to eliminate potential redacted texts (Sec. 2.1.1).
- Any non-redacted glyph shifts conditioned on redacted glyphs can be used to eliminate potential redacted texts (Sec. 2.1.3).

Where appropriate in future sections, we also address concerns related to nonexcising redactions. These redactions are cases where the text underneath the redaction can be selected and copied to the system clipboard from the PDF document.

2.1 Glyph Shifts

The width of a PDF redaction depends on glyph shifts. Without accounting for glyph shifts, redacted text guesses are imprecise and must account for error, reducing the potential of finding a unique match for redacted content. See the discussion of rasterization error in Section 7.1 for a precise measurement of how much shift error affects leaked redaction information.

The glyph shifts present in a PDF document are dependent on the specific *workflow* used to produce the PDF document. This includes an originating software, called the *PDF producer* by the

[1] We have released code for performing this conversion at [anonymized for peer review].

[2] For example, in the case of a TJ operator, the actual *glyph shift* includes any offset due to a positional adjustment as *part* of the calculation of its value.

[3] PDF offers the ability to redefine the text space. Edact-Ray accounts for this.

Table 1. Glyph width equivalence classes for the specific default version of Times New Roman used by Microsoft Word. Each number is the width in given to the glyph by the font file and each set of letters has equivalent widths.

569	ijlt	1251	ELTZ
683	Ifr	1366	BCR
797	Js	1479	ADGHKNOQUVXYw
909	acez	1593	m
1024	bdghknopquvxy	1821	M
1139	FPS	1933	W

ISO 32000 PDF standard [19], and any software that may modify the PDF file contents thereafter, including, for example, a redaction tool. A given workflow creates a specific pattern of glyph shifts, determining, in part, the security of any redacted text. We identify two types of glyph shifting schemes:

- *Independent*: the glyph shifts for a given character are not dependent on any other character in the document in any way.
- *Dependent*: the glyph shifts for a given character are dependent on some other character in the document in some way.

We call independent schemes *unadjusted* when there are no shifts on any character. Google Docs' *Export to PDF* option produces an unadjusted scheme.

The frequency of documents with a given shift scheme varies by corpus (Sec. 6). For example, in our Freedom of Information Act (FOIA) corpus about 6% of redacted text fragments are unadjusted, in contrast to 5.3% across all corpora.

2.1.1 Equivalence Classes. Before discussing these schemes further, we introduce the idea of width and shift equivalence classes. A shift equivalence class is a set of lists of glyphs of the same length with identical shift values. A width equivalence class is a set of glyphs and associated shifts with the same width.

Table 1 gives an example of the width equivalence classes for glyphs in a Times New Roman font[4] without shifts, next to their widths as specified by the font file. The glyphs *I*, *f*, and *r* are exactly half the width of the glyphs of *B*, *C*, and *R*. When typeset using Times New Roman, the words *martian*, *templar*, and *mineral* all have the same width, as do the anagrams of those words *tamarin*, *trample*, and *railmen*.

The PDF specification does not include any specific signifiers for redacted text. However, residual specification information after redaction, such as glyph positions, can be used to reasonably rule out large numbers of candidate width and shift equivalence classes for redacted text. None of the prior words in this paragraph are in the width equivalence class of the word *cat*.

2.1.2 Independent Schemes. In an independent glyph shifting scheme, the security of a redaction may be considered dependent on the size of the width equivalence class indicated by the PDF document's residual glyph positioning information. That is, the positions of glyphs prior to and succeeding the redaction may leak the width of redacted text. The scheme's specific glyph shifts can make a given width equivalence class leak more or less redacted information by making width of individual glyphs more or less unique.

As an example, consider redacting a single letter *l* as opposed to *m* in the TNR scheme from Table 1. *m* is the only glyph in its width equivalence class, so it may be possible to determine the letter *m* was redacted uniquely. Whereas if *l* is redacted, then the residual information indicates the

[4]Different versions of a font can exist. We use the default versions available from Microsoft throughout this paper.

redacted letter could be any one of *i*, *j*, *l*, or *t*. However, if *l* were always accompanied by a glyph shift distinguishing it from these other three letters, then the scheme would leak more information.

We acknowledge that there is no *guaranteed* correlation between glyph positions before and after redaction. Redaction may reposition glyphs in such a way as to destroy accurate width information. However, in Section 3 we find this is almost always the case for commonly accessible redaction tools. We have also been informed that in some contexts there are (legal) restrictions on changing the glyphs or glyph positioning of a redacted document.

Scanned Documents. Many documents with independent shifting schemes are the result of an optical character recognition (OCR) process.[5] This process embeds a non-raster representation of the document's text in the resulting PDF document. This often allows the document's text to be searched and copied from by software tools.

We note that attacking the OCR overlay is not always as straightforward as attacking a PDF produced without OCR. For example, attacking a redaction performed on the output of Adobe Acrobat Pro's OCR workflow requires using two side channels embedded in the PDF specification of the redacted document, detailed in Appendix F. Using these side-channels, we find documents processed with OCR using Acrobat Pro and then subsequently redacted are vulnerable to the attacks presented in this paper.

2.1.3 Dependent Schemes. A dependent scheme is more dangerous to the security of redacted text than an independent scheme. In these schemes non-redacted glyph shifts can be dependent upon redacted glyph information, because the non-redacted glyph shifts can be determined *before* redaction (Sec. 3).

Microsoft Word "Save as PDF". In this work, we focus on a class of dependent schemes defined by the Microsoft Word software's *Save As PDF* command.[6] We reverse-engineered the glyph shifting scheme produced by this command in Microsoft Word for Windows desktop versions 2007 to 2019. This process took around several months. Word 2007 to 2016 use one scheme, and Word 2019 to present versions use another. These two schemes affect thousands of real world document redactions (Sec. 6).

The studied dependent schemes accumulate a What You See Is What You Get (WYSIWYG) error measurement for each glyph from left to right across each line of text. If redacted content is not removed from a Word document before running "Save as PDF", redacted glyph positioning information affects the accumulated error value. *Thus information about the content of a redaction is leaked into the shifts applied to non-redacted characters.*[7] A majority of the redaction tools from Section 3 preserved all non-redacted glyph shifts.

Figure 2 depicts this behavior. Word's internal representation of the layout of characters for display purposes does not exactly match that of the TrueType Font (TTF) embedded in the PDF document. Word corrects for this small error between the two formats through use of glyph shifts. Suprisingly, these modifications are done independent of the user's screen resolution.

We note the internal widths used on line 3 of Figure 2 are determined by a loop with no overflow reset and the redacted information held by the accumulator *is not zero* after a single shift is written. We refer the interested reader to Appendix Figure 5: the "pixelWidths" variable in this figure is scaled by a constant to produce the "internalMSWordWidths" in Figure 2.

Word's shifting scheme depends on the document's edit history. Changing a character inside of a Microsoft Word document splits the internal representation of the text fragment containing the character into two fragments. This fragmentation resets the accumulation of glyph width error and

[5] We discuss physical document scanners in Appendix H.1.

[6] Shifting schemes generated by other workflows are detailed in Appendix Table 8.

[7] Different text justifications can leak *more* information. For an extended discussion see Appendix G.

```
for (int j = i + 1; j < vs->size(); j++) {
    t = ttfScaledWidths[j] / 1000;
    d = internalMSWordWidths[j] / internalMSWordFontSize;
    ttf += t;
    msWord += d;
    disp = ttf - msWord;
    if (
        ((disp > 0.003) || (disp < -0.003)) &&
        i != vs->size() - 1
    ) {
      int adj = disp * 1000 + 0.5;
      vs->setShift(j, adj);
      ttf = msWord = 0;
    } else {
      vs->setShift(j, 0);
    }
}
```

Fig. 2. Snippet of reverse engineered code representing how Microsoft Word leaks redacted character information into non-redacted characters in a PDF document.

affects the shifts emitted for a line of text. Edact-Ray accounts for this by considering all potential edits to redacted text.[8]

We validated our Word model on hundreds of lines of Wikipedia text rendered to PDF by Word, several manually-crafted test cases, and text fragments from redacted documents found in the wild. We give detail the Microsoft Word shifting scheme algorithms further in Appendix G.

We have submitted a bug report reporting these findings to Microsoft and have received a response indicating they are aware of the problem. While Microsoft is not *liable* for this vulnerability, removing the leaks from new versions of Word will improve the security of future redacted content.

3 Redaction Tools

We surveyed redaction tools to determine how much redacted text information they leak. Our results are in Table 2. To find these tools, we searched the web for the terms "PDF redaction tool," "redaction tool," and "document redaction tool". We then downloaded all redaction tools listed in the first five result pages. These tools represent what the typical user may encounter when searching for and using redaction software.

We find the redaction security vulnerabilities detailed by this paper are affected by redaction tools in one of four ways:

Full Text. The first family of tools draws a black box over text selected for redaction. The original PDF text object is still present and users can select and copy the text behind the black box. Two tools that fall into this category are listed as leaking *full text* in Table 2. We consider this to be a severe vulnerability and reported this to their developers (see Sec. 7).

Width and Shifts. The second family of redaction tools, referred to as *shift preserving*, attempts to perform redaction while maintaining the document's vector format. When applied to text objects, these tools replace the redacted text with a shift of equivalent width so that text outside the redaction remains at the same position as in the original document. We note the default behavior

[8]We found edits have no significant impact on the efficacy of deredaction. The information leaked is *at least* as much as an independent glyph shifting scheme. See Appendix G for the editing algorithm.

Table 2. Type of information leaked by redaction tools. Many tools leak redacted text width and glyph shift information. We include a star next to Adobe's tool. After discussion with Adobe, they claimed they do not use a raster approach. We found empirically this was not the case.

Redaction tool (flow)	*Version*	*Leaked information*
PDFzorro	N/A	Full Text
PDFzorro (lock)	N/A	Raster Width
PDFescape Online	N/A	Full Text
PdfElement (all)	8.3.10	Width and Shifts
Qoppa PDF Studio	2021.0.3	Width and Shifts
FoxIt PDF Pro (all)	11.0.1	Width and Shifts
Nitro PDF	13.58.0	Width and Shifts
Opentext Brava (all)	16.6.4.55	Width
Rapid Redact	2.3	Raster Width
redactpdf.com	N/A	Raster Width
Adobe (all)	21.5	Raster Width*
Adobe (postpone save)	21.5	Width and Shifts
IText PDFSweep	7.1.6	Width

of all these tools is to draw a black box over the redacted area.[9] Table 2 lists these as leaking *width and shifts*.

Width. The third family, Opentext Brava and IText PDFSweep, are listed as only leaking *width*:

Opentext Brava changes the underlying PDF font and shifts for non-redacted glyphs, however, the width of the original redaction is maintained to a close approximation and the number of characters in the redacted word leaks because every redacted character is replaced with a space character and an associated glyph shift.

IText PDFSweep considers the glyph shifts applied to each word, removes them, and applies their sum to the last glyph in the word before the redaction. This has the effect of maintaining the exact width of redacted text but removing information relating to how a specific glyph in a given word was originally shifted.

Raster Width. The last family of tools, referred to as *rasterizing tools*, starts by converting every document page into a raster image and then blacks out those pixels of the image that fall inside the area selected for redaction. Because pixels outside the redaction area are untouched, it is possible to recover the approximate width of redacted text from the spacing between glyphs on either side of the redaction box (Sec. 7.1). These tools are listed in Table 2 as leaking the *raster width* of the redacted text.

4 The Edact-Ray Tool Suite

Following our analysis of redaction tools, it was necessary to develop the Edact-Ray tool suite in order to better understand and fix the problem of deredaction. Edact-Ray automates the location, analysis, and protection of vulnerable redactions.

Locating Redactions. Edact-Ray adopts different approaches for identifying nonexcising and excising redactions.

For nonexcising redactions, Edact-Ray first detects filled boxes drawn over text. This algorithm has a large number of false positives as it does not consider whether the rectangles are actually

[9]This is typically user configurable. However, there are also often legal and organizational requirements on the style of redactions.

covering text in a meaningful way. A benefit of this approach is that it is fast: we use it as a first pass for identifying vulnerable nonexcising redactions in Section 6.

To avoid false positives, Edact-Ray then checks whether the pixels in the bounding box of each PDF glyph are all the same color. This does not handle complex cases (e.g. using an image to redact text), but we did not find cases of these more complex styles of redaction in practice.

We manually validated our nonexcising redaction location algorithm on PDF documents from RECAP, a website hosting US court documents, and discovered a false positive rate of 4%. An additional 7% of these documents properly changed the underlying text (e.g. to REDACTED) or contained unintended redactions (e.g. a black box covering non-sensitive text). This resulted in 710 documents with nonexcising redactions, many of which included entire paragraphs of text.[10] The algorithm encounters false negatives when it cannot identify rectangular draw commands. We did not encounter false negatives: redaction draw commands are rarely ambiguous enough to prevent detection.

For excising redactions, Edact-Ray first identifies spaces larger than a single space character[11] between each pair of words in a document. It then analyzes the pixel color values between the words to identify a drawn rectangle.[12]

We evaluated our excising redaction location algorithm's accuracy on a random sample of 1,000 corpora pages from Section 6. We compared against Tim B. Lee's [25] prior work on redaction location and against manual identification. Lee's method flags every black rectangle drawn as a redaction and has a high false positive rate for some classes of documents (around 30% for FOIA and nearly 100% for RECAP). With respect to false positives and false negatives, our algorithm is equivalent to manual analysis when locating what we deem to be vulnerable redactions.

The above approaches present a novel contribution of the present work. We collaborated with the Free Law Project on their tool x-ray [13], for identifying redactions. When we began collaboration, they communicated to us that there was no efficient tool with low false positives for identifying excising and nonexcising redactions. We have created such a tool, however, we note the purpose of this paper was not to solve the problem of redaction location.

Analyzing Redactions. Edact-Ray analyzes the security of a given redaction by developing a information fingerprint (i.e. a hash of the width and shift equivalence classes) of all leaked glyph positioning information. For each attempted guess at the redacted content, Edact-Ray matches the expected information fingerprint for this guess to the fingerprint recovered from the source document. Attempted guesses are selected from a *dictionary*.

If the guess information fingerprint does not match, that guess is ruled out as a potential redacted content. Edact-Ray's results are only as good as the dictionary. If the redacted text is not in the dictionary, Edact-Ray may return incorrect or no results. Deredaction does not *discover* the redacted text.

To minimize the chances of not including the correct redacted content in our guess dictionary, in the following sections we use a dictionaries consisting of multiple variations of possible redacted terms. We also performed extensive validation of our results wherever possible.

As a proof-of-concept tool, Edact-Ray relies on a human user to select the proper dictionary. While this process could be automated using machine learning, we chose to do this step manually in order to exclude a potential source of error.

Running on an eight-core consumer laptop (a ThinkPad T420), Edact-Ray eliminates around 80,000 guesses per second.

[10] We provided a list of these PDFs and associated case names to the US Courts.

[11] The tool is tuned for English language redactions for a number of reasons, chiefly the fact that the English alphabet has proportional widths.

[12] We did not attempt to locate redactions without some visually signifying feature.

Protecting Redactions. Edact-Ray protects vulnerable PDF redactions by first locating the nonexcising redactions and removing their underlying text from the PDF. We then adopt a user-configurable level of information excisement by allowing users to optionally remove all non-redacted glyph shifts[13] and optionally convert the font to a monospaced one, scaling the size to preserve readability. To protect excising redactions, Edact-Ray can round up the size of all spaces between two words to some width, $n \times w$, where n is some number of characters and w is width of a single character in the monospace font. Edact-Ray can also remove any rectangular draw commands from the PDF so that the width of the redaction cannot be recovered by examining the width of any graphical box drawn to represent the redaction.

While these changes may be unacceptable for many users, they guarantee the redactions' security in the general case. Monospaced redactions leak less than five bits of information (Sec. 5). We discuss further defenses and recommendations in Section 7. We have made code for implementing the above protections open source.

5 Measuring Leaked Information

We next demonstrate exactly how dangerous glyph shift aware redaction attacks are. We evaluate three types of glyph shifting schemes (Sec. 2): a monospaced scheme, a scheme with independent (unadjusted) shifts,[14] and two schemes with dependent glyph shifts (MS Word schemes).

We can quantify the amount of information leaked by glyph positioning in information theoretic terms, as the *mutual information* of the redacted text and the glyph positioning, measured in bits [21]. Mutual information measures the reduction in uncertainty about the redacted text from knowledge of glyph positions of the surrounding text (the *fingerprint* or *hash* of leaked information).

5.1 Experiment Setup

Text corpus. We estimate the amount of leaked information by simulating redaction on text from the New York Times Annotated Corpus [33]. The corpus contains articles written and published by the New York Times between January 1, 1987 and June 19, 2007. The NYT corpus contains 1,855,658 documents in total, with 1,027,427,021 total tokens, 1,505,676 of which are unique.

We considered each word and string of punctuation to be a separate token, excluding contractions. For example, a period and quotation mark (.") ending a quotation would be a token. J. Doe is tokenized to "J", ".", and "Doe". However, contractions such as "we're" are *not* tokenized into *we*, *'*, and *re*.

Dictionaries. The amount of information leaked also depends on prior information about the redacted text. For example, if we know the redacted text is one of 151,671 American surnames, then this redaction leaks at most $\log_2 151,671 \approx 17.2$ bits. In our experiment, we model this prior information as a *dictionary*, a set of strings from which we assume the redacted text is drawn.

Based on an examination of real redactions in Section 6, we constructed 10 dictionaries. We list the sources of these dictionaries in Appendix B.

- *Str.* All strings of 3–16 characters in length starting with a uppercase or lowercase letter followed by lowercase letters.
- *Acrn.* All strings of 2–5 uppercase characters.
- *Word.* English words including some proper nouns.
- *Ctry.* Official and common names of countries.
- *Rgn.* Names of regions, a superset of *Ctry*.
- *Natl.* Nationalities, demonyms, and adjectives of regions and nationalities, sourced from lists on Wikipedia.

[13] Recall even if the redaction width is changed these shifts can leak information.

[14] We use an unadjusted scheme for simplicity: each font evaluated in this section can be considered its own independent scheme (Sec. 2).

Table 3. Dictionaries containing candidate texts used for evaluating deredaction. Stop words are excluded from the statistics.

Dict.	*Size*	$H_u(X)$	*NYT Occ.*	$H_e(X)$
Str	$9{\times}10^{22}$	76.3	791,209,093	11.8
Acrn	$1.2{\times}10^{7}$	23.6	4,674,379	10.0
Word	63,054	15.9	432,896,070	12.3
Ctry	566	9.1	3,905,371	5.9
Natl	509	9.0	4,081,019	5.4
Rgn	$2.8{\times}10^{6}$	21.4	33,636,150	10.6
FN	100,364	16.6	71,031,188	10.3
LN	151,671	17.2	97,551,697	13.1
FILN	$1.6{\times}10^{6}$	20.6	1,265,265	17.0
FI×LN	$3.9{\times}10^{6}$	21.9	2,139,713	17.0
FNLN	$8.9{\times}10^{6}$	23.1	3,650,063	19.6
FN×LN	$1.5{\times}10^{10}$	33.8	14,440,238	19.6

- *FN.* American given (first) names.
- *LN.* American surnames (last names).
- *FI×LN.* All combinations of a name initial followed by surname (*LN*).
- *FN×LN.* All combinations of a given name (*FN*) followed by a surname (*LN*).
- *FNLN and FILN. FN×LN* and *FI×LN* filtered to only include combinations of name and surname that appear in the voter registration databases of the three US states. North Carolina [15], Ohio [16], and Washington [18] were chosen based upon the availability of publicly accessible data.

Table 3 lists the dictionaries and their statistics. *Size* gives the number of entries in the dictionary. *NYT Occ.* gives the number of occurrences of words from the given dictionary in the NYT corpus. $H_u(X)$ is the uniformly distributed information theoretic entropy of the dictionary, given by $\log_2$ *Size*. For dictionaries of individuals' names (those with *FN* and *FI* prefixes), $H_e(X)$ is the entropy of the empirical distribution of dictionary words in the voter registration databases.[15] For the other dictionaries $H_e(X)$ is the entropy of the empirical distribution of dictionary words in the NYT corpus.

The voter registration databases contained $1.7{\times}10^{7}$ names total. Note that if this measure was used instead of *NYT Occ.*, FN, LN, FILN, and FNLN would be identical to the population size.

Columns $H_u(X)$ and $H_e(X)$ are an *upper bound* on the amount of information a document can leak about a redacted element of that dictionary. If the amount of information leaked is equivalent to these numbers, then every word in the dictionary can be uniquely identified by leaked information when redacted.

Compared to the brute force approach of using *all possible* name combinations, using voter registration databases makes deredaction highly precise. However, we do not use *FNLN* and *FILN* in the evaluation of Section 6 because we do not want to bias our results by assuming what state the person (whose name was redacted) votes in or whether they are registered to vote.

We do not include strings of numbers in our dictionary list as the glyph advance width and effect on glyph shifts is almost always identical for digits, making them effectively monospace, even if typeset in a variable-width font.

[15]We use the voter registration database frequency for individuals' names to avoid bias present when using the NYT corpus to estimate the frequency of a given name. For example, "Al Gore" is more frequent in the NYT corpus than the population.

Workflow. We simulate redaction in a PDF created with three shifting schemes from Section 2 (Unadjusted, Word 2007 and Word 2019 "Save as PDF"), in 10 point Times New Roman, Calibri, and Arial, the three most common fonts in our document corpus (Sec. 6.1). These fonts account for 71.6% of lines in the 40,000 documents in Table 5. We include Courier as an example of a monospaced font. The simulated PDF, formatted for US Letter size paper, is left-justified with 1-inch margins, giving a 468 point line width.

We use a 10 point font size for our experiments as an upper bound on the amount of leaked information. We report results for a 12 point font size in Appendix C. Because each 12 point line has fewer characters, the larger font size leaks a little less information overall.

5.2 Experiment Procedure

We parameterize each redaction with experimental parameters of shifting scheme, font size, font, and dictionary. We then perform the following steps:

(1) Choose some word from the corpus that occurs in the given dictionary.
(2) Format a PDF document containing the chosen word and surrounding text from the corpus using the parameterized font and shifting scheme.
(3) Replace the chosen word at the point with each word in the parameterized dictionary (either uniformly or according to the frequency distribution).
(4) Redact the dictionary word by replacing it with a glyph shift equal to its width.
(5) Record the leaked information fingerprint for this dictionary word.

In summary, we simulated the redaction of all possible dictionary words at a sample of occurrences of dictionary words in the NYT corpus. We then computed the amount of mutual information leaked by the redacted document Y about the redacted word X (See Appendix A).

In addition to mutual information, we calculated the probability that the leaked information can be used to correctly guess the redacted word. In the uniform distribution this is a random guess from the (typically quite small) set of matching candidate words, and in the frequency distribution we select the candidate with the highest frequency in either the NYT corpus (for *Str*, *Acrn*, *Word*, *Ctry*, *Natl*, and *Rgn*) or in the three state voter registration databases (for *FN*, *LN*, *FN×LN*, *FI×LN*, *FNLN*, and *FILN*).

5.3 Results

Table 4 reports on the vulnerability of excising redactions. The top half of the table shows the case where dictionary elements are drawn uniformly at random in step 3 above and the bottom half shows the case where dictionary elements are drawn according to their frequency of occurrence in the NYT corpus or, in the case of names, in the three states' voter registration databases. We have omitted rows for the *FN×LN* and *FI×LN* dictionaries from the bottom half because they are identical to the *FNLN* and *FILN* results, respectively.[16]

The left side of the table shows the mutual information of the dictionary and the resulting document. This is the number of bits of information leaked by the document about the redacted word. These should be compared to the total entropy of the corresponding dictionary given in Table 3.

The right side of the table gives the probability correctly guessing the redacted text using only the information available after redaction. This corresponds to success probability of a game in which a player knows the dictionary and tries to guess the redacted word based information in the

[16]We considered using the empirical distribution generated by the independent distributions of first and last names in the voter registration databases. However, the distribution of first names is *not independent* of the distribution of surnames due to cultural naming conventions.

Table 4. Number of bits leaked (left) and probability of a correct guess (right) for different shifting schemes in simulated redactions of the NYT corpus set in 10pt font. Refer to Table 3 for the total number of bits of information present in the candidate dictionary. "Probability correct guess" refers to the likelihood of randomly selecting the redacted word given the (typically small) set of matching candidate texts.

		Leaked information (bits)									Probability correct guess								
Distr	Courier	Times			Arial			Calibri			Times			Arial			Calibri		
Dict	Mo	Un	W07	W19	Un	W07	W19	Un	W07	W19	Un	W07	W19	Un	W07	W19	Un	W07	W19
Uniformly distributed																			
Str	0.2	8.2	—	—	8.8	—	—	12.1	—	—	<1%	—	—	<1%	—	—	<1%	—	—
Acrn	0.2	6.5	11.0	9.2	6.4	10.9	9.7	11.4	13.9	13.7	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
Word	3.3	8.7	12.6	12.3	8.7	12.5	11.8	12.8	14.3	14.3	2%	22%	19%	2%	23%	16%	16%	48%	47%
Ctry	5.0	8.6	9.0	9.0	8.6	8.9	8.9	9.1	9.1	9.1	77%	94%	93%	75%	90%	89%	97%	98%	97%
Rgn	4.1	10.7	15.0	14.6	10.2	14.3	13.6	14.1	16.8	16.7	2%	10%	8%	1%	9%	7%	3%	15%	14%
Natl	3.8	8.2	8.8	8.8	8.0	8.8	8.7	8.9	8.9	8.9	66%	90%	91%	61%	90%	88%	97%	98%	98%
FN	2.6	7.8	11.6	11.1	7.9	11.0	10.4	12.3	14.1	13.8	<1%	11%	7%	<1%	10%	6%	8%	32%	28%
LN	2.9	8.2	12.4	11.7	8.3	12.2	11.4	12.7	14.8	14.6	<1%	11%	8%	<1%	12%	7%	6%	33%	30%
FILN	2.9	8.6	13.3	12.6	8.7	13.0	12.2	13.0	15.6	15.5	<1%	4%	2%	<1%	4%	2%	2%	11%	11%
FI×LN	2.9	8.5	13.1	13.1	8.6	13.2	13.2	12.8	15.4	15.3	<1%	1%	1%	<1%	2%	2%	<1%	5%	5%
FNLN	3.2	9.4	14.5	14.1	10.0	14.9	14.0	13.5	16.7	16.5	<1%	3%	2%	<1%	4%	3%	3%	8%	7%
FN×LN	3.4	8.8	13.7	13.3	9.2	13.8	12.9	12.8	15.9	15.3	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
Text frequency distr.																			
Str	3.0	7.6	—	—	7.5	—	—	10.5	—	—	37%	—	—	35%	—	—	74%	—	—
Acrn	2.0	6.5	8.7	7.7	6.4	8.1	7.8	9.1	9.5	9.5	44%	75%	59%	43%	67%	61%	81%	91%	90%
Word	3.2	8.1	10.9	10.6	7.9	10.6	10.2	11.2	11.8	11.7	29%	69%	63%	27%	64%	57%	74%	88%	87%
Ctry	3.0	5.6	5.8	5.7	5.6	5.7	5.7	5.8	5.8	5.8	92%	99%	96%	93%	97%	96%	97%	98%	98%
Rgn	3.1	7.4	9.3	8.9	7.3	8.9	8.6	9.6	9.9	9.9	53%	81%	75%	51%	75%	71%	88%	94%	93%
Natl	2.5	5.2	5.4	5.4	5.1	5.3	5.3	5.4	5.4	5.4	95%	99%	99%	90%	99%	98%	100%	100%	100%
FN	2.5	7.1	9.3	9.0	7.2	8.8	8.6	9.7	10.0	9.9	45%	79%	74%	46%	71%	68%	87%	93%	92%
LN	2.7	7.8	10.8	10.4	7.9	10.7	10.1	11.4	12.2	12.1	28%	59%	53%	28%	58%	51%	66%	81%	79%
FILN	2.7	8.4	12.4	11.9	8.5	12.2	11.5	12.6	14.4	14.3	8%	30%	22%	8%	25%	21%	34%	53%	52%
FNLN	3.1	9.2	13.8	13.6	9.8	14.3	13.5	13.2	15.9	15.8	4%	20%	19%	6%	24%	19%	16%	38%	37%

redacted document. In the non-uniform case, the optimal strategy is to guess the most likely (the highest occurrence frequency) word or phrase that produces the same redacted document.

The table gives statistics for four fonts: Courier, Times New Roman, Arial, and Calibri. For each font, we show the amount of information leaked by an unadjusted shifting scheme (Un), text produced using Word 2007–2016 (W07), and Word 2019–2021 (W19) dependent glyph shifting schemes. For Courier, we omit the W07 and W19 columns as monospaced fonts behave identically in the unadjusted and Word cases.

Table 4 does not show results for the *Str* dictionary under the uniform distribution using Word schemes because simulating redaction of all $9{\times}10^{22}$ strings is prohibitively expensive. Results for the unadjusted positioning scheme were obtained without simulating redaction by exploiting the regular structure of the dictionary.

We compare redaction vulnerabilities across three categories, using the example of redacting a surname set in 10 point Times New Roman font to guide the reader:

Monospace (Mo). For monospaced font redactions, the residual information in the document after redaction reveals the number of characters in the redacted word. The Courier column in Table 4 thus tells us how much information is revealed by knowing the number of letters in the word. (For monospace fonts, which are always unadjusted, this is just the entropy of the distribution of word lengths.) For example, knowing only the number of characters in a surname leaks 2.9 bits

of information (out of 17.2) when guessing uniformly at random from the candidate redacted texts. This has a $< 1\%$ probability of success—redactions of monospace fonts are relatively secure.

Unadjusted (Un). For independent shifting scheme redactions, e.g. a PDF produced by Google Docs, the residual width of the redaction leaks more information than the number of characters redacted. The width of a redaction of a surname (the *Un* row) with no glyph shifts provides 8.2 bits of information about the surname if it is chosen uniformly at random and 7.8 bits (out of 13.1) if it is chosen according to an empirical distribution. While the uniform distribution still has a $< 1\%$ probability of success, the empirical distribution has a 28% probability of success.

Dependent (W07/W19). For dependent shifting scheme redactions, the residual information after redaction leaks both width and shift equivalence classes (Sec. 2.1.1) and therefore more information. If we redact a surname from a PDF produced by Word 2007–2016, the resulting document leaks 12.4 bits of information about a name chosen uniformly at random and 10.8 bits when chosen according to the empirical distribution given by voter registration databases. With the inclusion of these information leaks, the probability of a correct guess under the uniform distribution is 11% and the probability under an empirical distribution is greater than 50%.

We found leaks of up to 15 bits of information about redacted text in dependent (Microsoft Word "Save as PDF") glyph shifting schemes. These schemes present a significant security concern for excising redactions. Even without considering the frequency of names in the population, single word redactions have a greater than 5% chance of being broken. When an adversary can use statistical likelihoods of names, two word redactions of first names and surnames face a 1 in 5 chance of being broken, and an adversary's chances are only better for other fonts.

Calibri, the default font in Microsoft Office since 2007, leaks the most information because the font has a greater variety of character widths than Times New Roman or Arial. Recall that the empirical entropy of the surname dictionary is 13.1 bits (Table 3), so a redacted surname set in 10-point Calibri using a Word 2007 shifting scheme leaks almost all the information available and can be correctly deredacted in roughly 81% of cases.

6 Redaction in the Wild

We next consider the question of whether these findings are applicable to real documents. This is not immediately clear because glyph shifts may be modified by a variety of software workflows.[17] It is also unclear whether there exist a significant number of vulnerable redactions in real documents.

We consider both *nonexcising* and *excising* redactions in this section. As mentioned in Section 1, nonexcising redactions retain redacted text in the PDF and only *visually* obscure the text. Excising redactions, on the other hand, remove the text from the PDF file.

We chose to study redactions of names (e.g. first names, surnames, and country names) in this section because they are the most common, discussed further below, and because their release presents a privacy concern. We do not release any of these names and have taken steps to notify affected parties in order to ensure no individual will be adversely affected by the present analysis (Section 7.3).

6.1 Experiment Setup

Corpora. We use the following real world document corpora:

(1) *FOIA.* Documents obtained via the US Freedom of Information Act (FOIA) on governmentattic.org [12]. This corpus provides us with independently selected documents with some public interest.

[17] For example, by opening the PDF produced by word *and then* modifying it using Adobe Acrobat. We perform an analysis of a few different workflows in Appendix E.

(2) *OIG.* Office of the Inspector General (OIG) reports hosted by oversight.gov [4]. The OIG is a US Government oversight branch tasked with preventing unlawful operation of other government branches. This corpus allowed us to measure the impact deredaction may have on documents from a high-profile and large organization.
(3) *DNSA.* Digital National Security Archive (DNSA) documents produced after 2010 [7]. The DNSA is a set of historical US government documents curated by scholars. That is, we found redaction information leaks affect significant historical documents.
(4) *RECAP.* CourtListener's RECAP court document archive. RECAP mirrors PACER, the US Federal Courts' docketing system [11], and contains over 10 million documents. We use RECAP to measure the impact of nonexcising redactions (discussed below).
(5) *rRECAP* the subset of RECAP documents returned for the search string "redacted". We chose to include rRECAP because running the excising redaction location algorithm mentioned in Section 4 on the entire RECAP corpus would be both computationally and financially prohibitive.

Only the RECAP corpus contained nonexcising redactions and our results for this corpus are reported with respect to nonexcising redactions. Our results for all other corpora are reported with respect to excising redactions.

Dictionaries. We chose to restrict our evaluation to first name and last name redactions, which are very common in our corpus. In a sample of 100 redactions from our corpora, 52 were personal names (determined manually based on the surrounding text), 26 were multi-word phrases too long to attack, 17 were numbers (not vulnerable to attack), 3 were pronouns (trivial to attack), and 2 did not have identifiable semantics. We composed a dictionary by taking the union of the first name and last name sets introduced in Section 5.

We also include titles, such as "Mr." and "Mrs.", as well as initials, such as "J." and "S.", in our dictionary. However, our results for matches *remove* these prefixes, so "Ms. Doe", "J. Doe", and "Doe" count as a single match.

Our final dictionary contained 7,066,800 entries. We do not include $FN \times LN$ as testing this dictionary takes 6 hours on an Intel Xeon Silver 4208, 2.10 GHz, 32-core server and the result sets are typically large.

We exclude the DNSA from this dictionary restriction. The redactions matching the Microsoft Word dependent glyph shifting scheme in the DNSA were of an acronym, demonyms, and countries. We used these dictionaries when performing deredaction on the DNSA (see Appdx. B).

Workflow. In this section we evaluate a redaction if either:
(1) The redacted text is present in the PDF (vulnerable to copy-paste attack); or
(2) The redacted text is not present, but the document retains glyph shifting scheme information where:
- The scheme matches a Word "Save as PDF" shifting scheme,
- The redaction appears to be a name, e.g. "Jane", and
- The redaction is the first from left to right on the line of text.

These criteria provide a uniform and accurate proof-of-concept evaluation of the impact of the discovered redaction vulnerabilities.

Limitations. We chose to restrict our attacks to the Microsoft Word schemes because these leak the most information, and are thus of the greatest concern. Other shifting schemes exist and evaluating these schemes' security will require further reverse engineering. Cases matching the Microsoft Word "Save as PDF" workflow represented 8.8% of redaction instances across all four corpora.

Section 2 noted Microsoft Word's glyph shifting scheme leaks redacted information from left to right. We therefore considered only the first redaction on a line to remove any dependence on prior correct guesses, which would be necessary to attack the second redaction on a line.

The cases of redaction we report are interesting and chosen conservatively. In reality, there may be additional vulnerable redactions not included in our count.

6.2 Experiment Procedure

We take three steps to ensure the results of our experiments are accurate:

(1) We require all potential deredactions to exactly match all present PDF glyph positioning information.
(2) For excising redactions, it is necessary to identify the shifting scheme used by the PDF document. This process avoids incorrect matching and filters out regions of documents with idiosyncratic formatting. For example, Microsoft Word documents may have alternative layouts for text, e.g. text boxes, tables, and line numbers.
In the present analysis, we consider a PDF page to *match* a scheme if we identify greater than 100 glyphs with shift values matching the scheme on the page. We only count entire lines: all of a line's glyph positioning information must match the scheme exactly to count toward the 100 glyph threshold.
(3) We manually classified each excising redaction as being a redacted name based upon the content of the surrounding text. We were conservative with our classification and only attack redactions we are *certain* are of names.
(4) Where possible, we validated our findings using public information.

Although not critical to our results, we also identified redactions in documents using unadjusted and Adobe OCR glyph shifting schemes. In Section 5, we found these schemes leak 2–3 bits less information than documents produced by Microsoft Word.

To make sure we were missing any PDF documents originating from Microsoft Word, we also counted redactions within a 10 text space unit L_1 distance from our Microsoft Word model. These PDFs were the result of a non-standard workflow, e.g. creation using Word 365, the web interface for Microsoft Word. In our results, we label these redactions as *Near Word*.

6.3 Results

Excising Redactions. Table 5 reports our findings on the security of excising redactions. The *vulnerable* row reports the result of our redaction classification methodology. We identified 711 vulnerable excising redactions in the FOIA corpus, 58 in the OIG corpus, 9 in the DNSA corpus, and none in rRECAP (mentioned above). The lower half of the table reports statistics on the number of entries from our dictionary that were not determined to be impossible given observed leaked redaction information.

Due to our large combined dictionary size, we saw few unique matches (0.4% of 769 FOIA and OIG redactions). Deredaction still presents a serious threat, particularly in the case of constrained dictionaries. For example, knowing a redaction is a U.S. congressperson's name would allow an attacker to deredact the name.

The size of the possible redacted texts after redaction is also skewed positively. While FOIA's average case sees around a 3,000-fold reduction in the number of potential redacted texts, the majority of cases see at least a 14,000-fold reduction.

We find unmatched cases are common: false negatives arise if the word does not occur in the dictionary. Manual analysis of nonexcising RECAP redactions found 28.2% of names were of a form occurring in our dictionaries, leaving 71.7% in other forms (e.g. first name, last name). Our own

Table 5. Top: Glyph shifting schemes identified in redacted corpora pages. Bottom: Deredaction results for names tagged.

Metric	FOIA	OIG	DNSA	rRECAP	RECAP
Documents	3,145	1.9×10^4	678	2.5×10^4	$\approx 10^7$
Pages	4.9×10^5	5.2×10^5	1.2×10^4	3.4×10^5	$\approx 10^8$
Redacted PDFs	236	1255	7	67	710
Redactions	4.5×10^4	1.3×10^4	235	1,221	6,541
Unadjusted	2,844	314	7	7	327
Adobe OCR	3,406	1,814	0	224	445
Near Word	175	114	3	33	20
Unrec.	1.3×10^4	1×10^4	214	838	5,691
Exact Word	4,694	455	14	119	58
Vulnerable	711	58	9	0	58
No Matches	382	39	1	–	N/A
Uniq. Matches	3	0	5	–	N/A
Avg. Matches	2,435	4,260	393	–	N/A
Med. Matches	494	1,081	1	–	N/A

false negative rate (54.7%) is lower because, while we extensively tested our positioning scheme models, misleading results (false positives) appear to exist in approximately 17% of cases.

Nonexcising Redactions. We ran Edact-Ray's location algorithm for nonexcising redactions on all of RECAP and found 6,541 nonexcised redacted names in US court documents. The number of nonexcised redacted words was larger ($\approx 1.4\times10^5$).

6.4 Validation

We manually validated our results for excising redactions (where possible) by performing web searches for further information on the redacted document. For example, in the case of an OIG investigation, we would perform a search related to the offense committed and organizations affiliated. This process found our deredaction returned no false positives, though our false negative rate suggests some of our match sets may not include the redacted word. False positives occur when Edact-Ray returns a set of matches, but the redacted name is not in the dictionary. Given the danger of false positives, deredaction should be understood as *ruling out* possible candidate texts rather than determining the content of a redaction.

Nonexcising redactions provide ground truth, and we also used these redactions to validate our techniques. For every nonexcised redacted name in RECAP present in our dictionary, we excised the name, i.e. we removed the redacted text but preserved non-redacted glyph shift information. We then formed a set of matching names using Edact-Ray, and we were successful in all cases, as the set of candidate texts returned by Edact-Ray included the ground truth redacted word.

Two of these nonexcising redactions were of a name only (no other words on the line were redacted). We report results for these redactions as Edact-Ray's penultimate evaluation:

Mr. Hamilton. The first redacted name had the form *Mr. Hamilton* (actual name different). The result set contained 24 names and Mr. Hamilton's was the 14th most common based on US Census data. No matched name had the title *Ms.*.

Ms. Schuyler. The second redacted name had the form *Ms. Schuyler* (actual name different). The result set contained 210 names and Ms. Schuyler's was the 127th most common based on US Census data. No matched name had the title *Mr.*.

7 Discussion

In the prior sections, we demonstrated sub-pixel sized glyph position shifts, imperceptible to the human eye, can break text redactions. We found no non-rasterizing redaction tools address these leaks and two of the tools, PDFescape Online and PDFzorro, do not even excise redacted glyphs. There are at least 778 vulnerable excising redactions in FOIA, OIG, and DNSA PDF corpora and more than 700 publicly accessible court documents with nonexcising redactions.

We have provided a list of the nonexcising redactions and associated case numbers to RECAP and the US Court system. We also notified other affected parties of the leaks and provided them with the tools to fix them. In particular, our results show redacting a name from a PDF is not secure and new redaction practices should be adopted (Sec. 7.2).

7.1 Defenses

Excising redactions may be safeguarded. The official NSA guidelines for redaction of Microsoft Word generated PDFs are to change the content of the original Word document so that information regarding the sensitive name is destroyed (i.e. by changing the name to the letter "x") [2]. Below, we describe five alternative defenses against excising redaction vulnerabilities. We are releasing parts of the Edact-Ray tool suite that can be used to identify non-excising redactions, and tools to repair such redactions. We are *not* releasing the Word document models that can be used to attack vulnerable excising redactions.

Glyph Shift Discretization. Shifting scheme noise may be added to a line without affecting visual fidelity.[18] Alternatively, each shift could be rounded to a discrete interval, e.g. 0.1 mm. If this noise is indistinguishable from legitimate glyph shift information, accounting for it would require an adversary to increase the set of accepted redacted text guesses. Removing shifting scheme information altogether can also lower information leaks, though this is less visually appealing.

Document Layout and Redaction Obfuscation. Modifying the PDF commands used to render the box of the redaction complicates the process of automated redaction location. For example, our nontrivial redaction location algorithm relies on identifying a black box between two US English words in order to avoid large numbers of false positives.

Increasing Redaction Width. Redacting additional adjacent words can make deredaction more difficult, although this practice may not always compatible with legal mandates. This defense works increases the number of words the attacker must guess. If the adjacent words are easy to infer, this is not a sufficient defense.

Adversarial Redactions. One defense against deredaction is to change the document's text before it is redacted, for example, by replacing a sensitive name with the letter "X". It is also possible to *lie* to deredaction by changing the redacted content to something seemingly valid, potentially misinforming an adversary.

Rasterization. Rasterization appears to be an effective defense against deredaction. In many cases this defense is infeasible because it removes searchable text data from the document, however, performing OCR on the document post-redaction can act as a stopgap for this issue. Rasterization algorithms may also modify or ignore certain glyph shifts,[19] requiring the analyst to perform more reverse engineering to identify the specific rasterization tool used.

We performed an experiment to estimate the effects of rasterization on redaction security. We chose ten name occurrences from court documents using the Word shifting scheme in 12 point Times

[18] Recall each text space unit is 1/6,000 in (0.0042 mm) (Sec. 2).

[19] We analyzed several rasterization tools, finding several had imperfect precision.

New Roman. For each selected text line, we substituted the name with each entry in our 235,560 name dictionary from Section 6, redacted the entry, and calculated the redaction's width. This resulted in 2,017 unique redaction widths, ≈11 bits of information leaked, on average. Quantization to 300 DPI (20 text space units) resulted in 252 widths on average (≈8 bits) and quantization to 600 DPI performed slightly worse with a result of 377 widths (≈8.6 bits).

Unfortunately, this estimation is a lower bound. In general, when a character is rasterized, the vector representation is converted to a "mosaic" of pixel values. Whether a given pixel value is black, white, or in the case of anti-aliasing [32], some shade of gray, is dependent on how the graphical rendering specification is converted to a matrix of pixel values. We leave an analysis of this precision loss due to particular rendering algorithms to future work.

7.2 Recommended Practices

Redaction practices must account for concerns about *document integrity*. All the above measures modify the redacted document beyond simply removing text. In some contexts, particularly due to legal or regulatory reasons, this may not be acceptable. One of the main reasons for releasing redacted documents is to demonstrate *transparency* while still protecting sensitive information. Altering parts of the document outside the redaction alters this promise of authenticity.

It is technically possible to fix a non-excising redaction by removing the redacted text. The effect would be the same as if the document were redacted by an excising redaction tool. However, this also raises the issue of authenticity of done by a third party (e.g. document repository operator), because this necessarily means modifying the original document without the author's involvement.

In cases where integrity requirements may be relaxed, the NSA-recommended practice of altering the original document to replace the redacted text with meaningless text, e.g. REDACTED, provides the highest level of security.

In cases where the underlying text may not be changed, we offer the following two suggestions. First, we note that **redacting a name from a PDF is not secure**. If a name occurs on a line of text, the entire line should be redacted, if possible, or care should be taken to ensure that enough of the surrounding words are redacted to make deredaction unlikely. Second, if redacting more text is not possible, the width of the redaction should be quantized to a fixed value, and any glyph shifts should be removed. While this may make the file less aesthetically pleasing, it is necessary for the security of redactions.

7.3 Ethics

We carefully considered the balance between the benefits of public awareness and potential risks of misuse present in this research. To the best of our abilities we are attempting to maximize the benefits to society and minimize the harm to individuals in the publication of this work. In this spirit we have released open-source toolkits for protecting vulnerable redactions (Sec. 4).

All the documents studied are in the public domain. So long as copies of these PDFs exist, they pose a risk to individuals' privacy. During our evaluation, we ensured all deredaction was performed on an isolated and hardened server and that no identifying information exited this server. We deleted the documents from our server and associated data after evaluation and notifying affected parties.

We have notified Microsoft and Adobe of our discoveries: although we understand they are *not responsible* for the vulnerabilities, changes to their tools could help protect future PDF redactions. We have also reached out to the PDF Association regarding the provision of guidance for redaction application implementers.

We reached out to the two redaction toolkits with full text redaction leaks and have been assigned CVE-2022-30350 and CVE-2022-30351. We have also performed an extensive notification of the US

Courts, the Free Law Project, the US Department of Justice, and the Office of Inspector General. Our discussions with these groups are ongoing and include technical support, code, and free consulting regarding remedation and prevention efforts.

Careful deliberation has led us to determine that the present paper will motivate further remediation efforts rather than encourage attacks on redacted documents. By bringing attention to the problem of redaction, this paper empowers affected individuals to advocate for the protection of their redacted information.

8 Related Work

Our work is the first to consider the role the PDF glyph positioning information plays in redaction security. We are also the first to present an algorithmic attack on redactions where the underlying text is removed and replaced with a black box.

Digital Redaction However, we are not the first work to discuss digital redaction. Forrester and Irwin [23] discuss trivial redactions and unscrubbed metadata such as the Producer field of PDF documents but do not mention glyph positioning based deredaction. Hill et al., used hidden Markov models to recover text obscured either by mosaic pixelization or a related tactic, e.g. Gaussian Blur [24]. While Müller et al. [29] do not explicitly tackle redaction, they discuss hidden information present in PDF documents, specifically PDF document revision information and author name metadata. Beyond PDF document redaction, other file formats may also be deredacted: Murdoch and Dornseif [30] discuss how cropped JPEGs can preserve uncropped image information.

Text Redaction Attacks The primary predecessor to our work is Lopresti and Spitz [27], which presents a manual technique for matching glyphs to a redaction's width in a raster image of text. The authors attempted to use natural language processing to predict redacted words, something our work found imprecise given current models. The Lopresti and Spitz work also conflates document glyph position specifications with TTF glyph widths and assumes both are equivalent to a raster document's character widths. This presents two problems:

First, a rasterization workflow may change a document's glyph positioning and physical printing may not be a pixel-perfect reproduction of the digital document. While the accurate representation of documents is the whole *point* of PDF, a few of our tests found glyph positioning operations were not always honored. Behavior and information from any *singular* tool may or may or may not comply with the PDF standard, either because of software bugs or because the developers were too lazy to implement support for a specific glyph positioning operation.

Second, TTF glyph widths do not necessarily equate with PDF document or raster glyph widths. TTF is only one of five types of fonts supported by PDF. The Lopresti and Spitz techniques also rely on (potentially inaccurate) human determination of glyph positions. Recall the present work provides a fully automatic deredaction method, a precise analysis of leaked information, and a clear measurement of this problem's prevalence in real documents.

Efforts by Government Organizations Outside the scientific community, some government agencies have studied redaction vulnerabilities. The Australian Cyber Security Center [3] analyzed Adobe Acrobat 2017's redaction security and considered several features including encryption, CMap leaks, redactions of text metadata, images, revision metadata, and form metadata. However, this work does not address glyph positioning information leaks and incorrectly determines that Adobe Acrobat leaks no redacted information. The National Security Agency's redaction guide [2] does not mention glyph positioning information but notes any underlying redacted text should be removed from the document before producing a PDF. Changing the underlying text before

redaction is a great defense against deredaction attacks, however, we found redactors do not always follow this advisory.

High Profile Broken Redactions There have been many other cases of poorly redacted, high-profile documents. Trivial redactions have been found in classified US Military documents [34], Manafort case documents [20], and documents relating to Larry Page's house [28]. The AstraZeneca Contract with the EU disclosed funding information in the PDF's bookmarks [31]. One Ghislaine Maxwell interview leaked redacted information on President Clinton in the document's index [26].

9 Conclusion

In the course of this work, we developed Edact-Ray, a tool capable of locating and breaking redactions in thousands of PDF documents. We found, for example, that redacting a surname from a PDF generated by Microsoft Word set using 10-point Calibri leaves enough residual information to uniquely identify the name in 14% of all cases. We surveyed the behavior of 11 different PDF redaction software tool-kits and found the majority do not defend against our attacks. We discovered over 6,000 nonexcising redactions in US court documents and broke 348 excising redactions in Office of Inspector General reports and Freedom of Information Act requests. This paper's findings are a lower bound on the extent of vulnerable redactions.

References

[1] Surnames Occuring 100 or More Times. https://www.census.gov/topics/population/genealogy/data/2000_surnames.html, 2000.
[2] Redacting with Confidence: How to Safely Publish Sanitized Reports Converted From Word to PDF. 2005.
[3] An Examination of the Redaction Functionality of Adobe Acrobat Pro DC 2017. https://www.cyber.gov.au/acsc/view-all-content/publications/examination-redaction-functionality-adobe-acrobat-pro-dc-2017, 2017.
[4] All Inspector General Reports in one Place, 2021.
[5] Beyond the Top 1,000 Names. https://www.ssa.gov/oact/babynames/limits.html, 2021.
[6] Demonym. https://en.wikipedia.org/wiki/Demonym, 2021.
[7] Digital National Security Archive. https://nsarchive.gwu.edu/digital-national-security-archive, 2021.
[8] List of Adjectival and Demonymic Forms for Countries and Nations. https://en.wikipedia.org/wiki/List_of_adjectival_and_demonymic_forms_for_countries_and_nations, 2021.
[9] List of Alternative Country Names. https://en.wikipedia.org/wiki/List_of_alternative_country_names, 2021.
[10] NGA GEOnet Names Server (GNS). https://geonames.nga.mil/gns/html/, 2021.
[11] Public Access to Court Electronic Records. https://pacer.uscourts.gov/, 2021.
[12] The Government Attic, 2021.
[13] X-Ray Bad Redaction Detector. https://free.law/projects/x-ray, 2021.
[14] LibreOffice Github. https://github.com/LibreOffice, 2022.
[15] North Carolina Voter Data. https://www.ncsbe.gov/results-data/voter-registration-data, 2022.
[16] Ohio Voter Data Download. https://www6.ohiosos.gov/ords/f?p=VOTERFTP:STWD:::#stwdVtrFiles, 2022.
[17] The TeX Live SVN. =https://www.tug.org/texlive/svn/, 2022.
[18] Washington Voter Registration Database Extract. https://www.sos.wa.gov/elections/vrdb/extract-requests.aspx, 2022.
[19] PDF Association. *ISO 32000-2*. 2017.
[20] Natasha Bertrand. Manafort's Own Lawyers May Have Hastened His Downfall. https://www.theatlantic.com/politics/archive/2019/01/paul-manafort-lawyers-failed-to-redact-documents/579910/, 2019.
[21] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 2 edition, 2006.
[22] Herbert Dixon. Embarrassing Redaction Failures. https://www.americanbar.org/groups/judicial/publications/judges_journal/2019/spring/embarrassing-redaction-failures/, May 2019.
[23] Jock Forrester and Barry Irwin. An Investigation into Unintentional Information Leakage through Electronic Publication. *Information Security South Africa*, 2005.
[24] Steven Hill, Zhimin Zhou, Lawrence Saul, and Hovav Shacham. On the (In) Effectiveness of Mosaicing and Blurring as Tools for Document Redaction. *Proceedings on Privacy Enhancing Technologies*, 2016(4):403–417, 2016.
[25] Timothy Lee. What Gets Redacted in Pacer? https://freedom-to-tinker.com/2011/06/16/what-gets-redacted-pacer/, 2011.
[26] Josh Levin, Aaron Mak, and Jonathan Fischer. We Cracked the Redactions in the Ghislaine Maxwell Deposition. https://slate.com/news-and-politics/2020/10/ghislaine-maxwell-deposition-redactions-epstein-how-to-crack.html, 2020.

[27] Daniel Lopresti and A Lawrence Spitz. Quantifying Information Leakage in Document Redaction. In *Proceedings of the 1st ACM workshop on Hardcopy document processing*, pages 63–69, 2004.
[28] Cade Metz. Privacy Watchdog Hoists Google by Its Own Petard. https://www.theregister.com/2008/08/01/nlpc_outs_larry_page/, 2008.
[29] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. Processing Dangerous Paths. NDSS, 2021.
[30] Steven Murdoch and Maximillian Dornseif. Far More Than You Ever Wanted To Tell: Hidden Data in Internet Published Documents. http://irc.smurfnet.ch/events/CCC/congress/21c3/papers/271%20Hidden%20Data%20in%20Internet%20Published%20Documents.pdf, 2004.
[31] Nikolaj Nielsen. EU Admits Redaction Error in AstraZeneca Contract. https://euobserver.com/coronavirus/150799, 2021.
[32] Olexander N Romanyuk, Sergii V Pavlov, Olexander V Melnyk, Sergii O Romanyuk, Andrzej Smolarz, and Madina Bazarova. Method of Anti-Aliasing with the Use of the New Pixel Model. In *Optical Fibers and Their Applications 2015*, volume 9816, pages 274–278. SPIE, 2015.
[33] Evan Sandhaus. The New York Times Annotated Corpus. https://catalog.ldc.upenn.edu/LDC2008T19, 2008.
[34] David Willey. Italy Media Reveals Iraq Details. http://news.bbc.co.uk/1/hi/world/europe/4504589.stm, 2005.

A Mutual Information Calculation.

The amount of information leaked by the redacted document Y about the redacted word X is given by the mutual information $I(X, Y)$. Let L be a random variable representing the location of the occurrence of the dictionary word chosen in our first evaluation step, and H denote entropy. Note that $H(Y|L, X) = 0$ (there is no uncertainty about Y given L and X), since the formatting and redaction in steps 3 and 4 are deterministic, and $H(L|X, Y) = H(L|Y) = 0$ (there is no uncertainty about the location of the redaction given the document after redaction). Using these two facts,

$$\begin{aligned}
I(X;Y) &= \\
&= H(X) - H(X|Y) \\
&= H(X) - H(X,Y) + H(Y) \\
&= H(X) - H(L,X,Y) + H(L|X,Y) + H(L,Y) - H(L|Y) \\
&= H(X) - H(Y|L,X) - H(L,X) + 0 + H(L,Y) - 0 \\
&= H(X) - 0 - H(L) - H(X) + H(Y|L) + H(L) \\
&= H(Y|L) \\
&= \sum_{\ell} Pr[L = \ell] \cdot H(Y|L = \ell).
\end{aligned}$$

Thus, $I(X;Y)$ is the average, taken over redaction locations, of the entropy of Y, that is, the entropy of the distribution of possible documents after redaction. For a large corpus and large dictionaries, calculating this quantity exactly is expensive. Instead of calculating the exact value, we sample $H(Y|L = \ell)$ for several initial word choices.

B Dictionary Construction

This paper used several dictionaries. In order to ensure the quality of the guessed text, we manually sourced and refined the following:

(1) *Word* was sourced from the linux spellchecker: /usr/share/dict/american-english filtered for pronouns (first letter upper case), stop words sans pronouns, and words containing punctuation and digits. Single quotes were made to be curly possessive and straight to match the document under study.
(2) *FN* were sourced from from the US social security agency [5] starting from 1880. For both this and the last name dictionary names like “McCarthy” were made to be correctly capitalized.
(3) *LN* were sourced from the year 2000 US Census [1].

(4) *Ctry* was sourced from wikipedia lists of countries and alternative country names [8, 9], then refined by hand to ensure completeness with repect to potential forms, e.g. United States vs. The United States. For fun we added intialisms.
(5) *Rgn* is from the NGA GEOnet Names Server (GNS) [10]. We filter for full name and BGN-approved local official name. We restrict our results to A, P,and L feature categories.
(6) *Natl* is sourced from wikipedia lists of demonyms [6, 8].

C Leaked Information for 12 Point Fonts

Table 6. Leaked 12 point font information. 12 point font results for unadjusted and monospaced schemes are the same as Table 4.

	Leaked information (bits)						Probability unique match					
Distr	Times		Arial		Calibri		Times		Arial		Calibri	
Dict	W07	W19	W07	W19	W07	W19	W07	W19	W07	W19	W07	W19
Uniformly distributed												
Acrn	10.5	9.4	10.5	9.5	14.2	14.1	<1%	<1%	<1%	<1%	<1%	<1%
Word	11.9	11.1	12.2	11.3	14.7	14.6	6%	3%	9%	5%	38%	37%
Ctry	9.0	9.0	8.9	8.9	9.1	9.1	88%	84%	83%	79%	95%	95%
Rgn	14.4	13.6	14.1	13.3	17.1	17.0	5%	3%	4%	3%	10%	9%
Natl	8.5	8.4	8.5	8.4	8.7	8.7	79%	74%	78%	72%	96%	96%
FN	11.0	10.1	10.9	10.2	14.2	14.2	3%	1%	3%	2%	18%	17%
LN	11.6	10.8	11.9	11.1	15.0	14.9	2%	1%	3%	2%	19%	18%
FILN	12.5	11.6	12.7	11.9	16.1	16.0	2%	1%	3%	2%	15%	14%
FI×LN	12.6	11.7	12.9	12.1	16.3	16.2	<1%	<1%	<1%	<1%	4%	3%
FNLN	14.1	13.1	14.4	13.6	17.1	17.1	3%	2%	3%	2%	10%	10%
FN×LN	12.7	11.8	12.8	12.1	16.9	16.9	<1%	<1%	<1%	<1%	<1%	<1%
Text frequency distr.												
Acrn	8.2	7.8	8.1	7.7	9.6	9.6	3%	3%	3%	2%	9%	9%
Word	10.3	9.8	10.5	9.8	11.9	11.9	4%	1%	5%	2%	38%	36%
Ctry	5.8	5.7	5.8	5.7	5.8	5.8	83%	79%	82%	73%	91%	92%
Rgn	9.3	9.0	9.1	8.9	10.0	9.9	1%	<1%	1%	<1%	5%	5%
Natl	5.3	5.3	5.3	5.3	5.4	5.4	78%	75%	78%	71%	97%	98%
FN	9.0	8.6	8.8	8.5	10.0	10.0	74%	67%	71%	66%	93%	93%
LN	10.4	9.8	10.5	9.9	12.3	12.3	54%	47%	55%	48%	83%	83%
FILN	11.9	11.1	11.9	11.2	14.7	14.7	25%	18%	23%	19%	58%	57%
FNLN	13.6	12.7	13.8	13.1	16.2	16.2	20%	15%	20%	17%	41%	41%

Overall, we found 12 point font sizes leak only slightly less information in dependent schemes than 10 point font sizes. However, the difference is not significant: we report our results in Table 6. For Calibri, the larger font size leaked *more* information, because the larger font size emphasizes the small errors accounted for by Word's glyph shifting scheme.

Note that we do not reproduce the independent scheme results for this table since they are identical to Table 4.

D Redaction Location

In Table 7 we report the comparison between our algorithm for locating excising redactions and the one provided by Timothy B. Lee [25]. We give an outline of this algorithm in Figure 3. This

Table 7. Accuracy of locating excising redactions using a box-walk routine vs. Timothy B. Lee's method, which records graphics state draw commands and looks for rectangles. Here we seperate redactions into "easy" and "hard" categories, where "hard" redactions are, for example, boxes drawn in the document with no surrounding text, or redactions that extend across and entire line and thus are too long to attack.

	DNSA	*FOIA*	*Govt.*	*rRECAP*
Box				
Easy Identified	9	84	61	0
False Negative Easy	0	3	0	0
Hard Identified	9	34	71	1
False Positives	0	0	0	0
Lee				
Easy Identified	1	2	25	0
False Negative Easy	0	0	0	0
Hard Identified	10	53	33	1
False Positives	0	6	26	57

algorithm, we note, is optimized to find boxes with respect to *other non-redacted words* on the page, whereas Lee's method looks for rectangle draw commands. Lee's method is therefore better at detecting redactions with no surrounding text, but these redactions are less useful for the deredaction attacks on excising redactions presented in this paper. We also added a second step to the x-ray algorithm [13] for locating nonexcising redactions which removed a large number of false positives from the results (algorithm in Figure 4).

These bash scripts call into C, python, and ruby for various subroutines. We do not use open source tools other than Poppler for this work, because in general we found most tools were inaccurate. We performed extensive validation (mentioned throughout this paper) to ensure our recovery of PDF glyph positioning information was precise and correct.

The "pts" command is our own, and handles lifting a PDF specification into an intermediate representation consisting of glyph coordinates at the scale of text space units, mentioned briefly in Section 2. find_redaction_box.py and trivial_redaction_loc.py work as described earlier in this paper. The former checks every sufficiently large space between two words for a redaction rectangle and the latter compares the pixels of the two rasterized images at glyph coordinates piped in by get_word_coords.rb.

```
tf=$(mktemp -u)
# Removes images from PDF
cpdf -draft "$1" -o "$tf".pdf-0
$SRC/painting/remove-tjs.sh "$tf".pdf-0 "$tf".pdf
convert -background 'rgb(255,255,255)' \
        -colorspace gray -normalize \
        -density 300 -quality 100 -depth 8 \
        "$tf".pdf "$tf".ppm 2>/dev/null
# parses out all the PDF text state and
# walks redaction boxes
$SRC/c-src/pts "$1" 1 |
    $SRC/location/find_redaction_box.py "$tf".ppm
```

Fig. 3. Nontrivial redaction location algorithm

```
# convert all text to black
"$DIR"/lib/camlpdf -blacktext "$1" -o "$1"-a
# remove all text from one pdf
gs -q -o "$1"-b \
   -sDEVICE=pdfwrite -dFILTERTEXT "$1"-a
# create two ppms
pdftoppm -singlefile -r 300 "$1"-a "$1"-a
pdftoppm -singlefile -r 300 "$1"-b "$1"-b
# get coordinates of each word and compare pixels for differences
"$DIR"/get_word_coords.rb "$1"-a |
  "$DIR"/trivial_redaction_loc.py \
  "$1"-a.ppm "$1"-b.ppm
```

Fig. 4. Two-pass algorithm for locating trivial redactions

E PDF Workflows

We performed an analysis of several PDF document production workflows across a variety of operating systems and software. Our results are reported in Table 8. This table demonstrates the wide variety of glyph shifting schemes in different software sets. Each row grouping represents a different general environment and software creator for the production of the PDF document, and each row represents a specific workflow—mostly which buttons are pressed during the PDF's creation.

F The Acrobat Pro Glyph Shifting Scheme

We can divide the effects of Acrobat Pro on PDF glyph shifts along into two functionalities: document editing and document OCR. Clicking on the text of an existing PDF in Acrobat does not change any positioning information. However, when a user edits a text object (typically one line of text) Acrobat sets all the object's glyph shifts to 0, making the scheme unadjusted. During OCR, Acrobat (1) creates text objects and (2) adds glyph shifts to otherwise unadjusted text. Acrobat's OCR algorithm creates glyph shift values by adding character and word spacing operators to each word, starting at the first letter of each word. The former, `Tc`, adds a small shift to each character in the word. The latter, `Td`, adjusts the word's x,y coordinates relative to the end of the prior word.

Redaction tools may remove these text positioning operators, making it impossible to infer the precise width of the redacted word. This is because the redacted text's width (as given by unredacted glyphs) may not necessarily be equivalent to the total glyph shift used to represent the redaction. However, we found these operators' parameters can be inferred via two sidechannels:

(1) **Tc**: This operator also applies to the successor space character of the word. Since this space is rarely redacted, the applied Tc command remains in the PDF after redaction. This allows us to infer the precise spacing applied to the redacted text's characters, and therefore precisely infer the width of redacted text when used in combination with the Td operator sidechannel (discussed next). We validated this for all redaction tools in Section 3.
(2) **Td**: All of Section 3's redaction tools draw a box after removing the glyphs for the redacted text. This black or white box's coordinates match those of the first glyph in the redacted word *after* the Td adjustment, revealing the redaction's exact width. If redaction removes the word's predecessor space character, this sidechannel is removed. There may not exist a space character before the redacted information, however, in this case other information such as the left justification margin may be inferred from other text on the page and used instead.

Table 8. Several possible PDF workflows. Each entry in the column on the left should be read from left to right, indicating the stages used to produce the document. For example, "Edge/Firefox, Print/PDFViewer, Save" can be interpreted as "using the edge or firefox browsers' PDF viewer or print dialog, hit save" and results in the right column's displacements.

Software Stack, Workflow	Description of Positioning Scheme
Word 365, Windows 10	
Edge/Firefox, Save	[(E)7(xhi)7(bi)7(t)7(A)-6(.)]
Edge/Firefox, Edited, Save	[(E)7(xhi)7(bi)7(t)7( )-1.75(A.)]
Edge/Firefox, Print/PDFViewer, Save	[(E)-13(xhi)28(bi)28(t)-34(A)-27(.)]
Edge/Firefox, Edited, Print/PDFViewer, Save	[(E)-13(xhi)28(bi)28(t)-34( )-3.17(A.)]
Edge, Edited, Print/PDFViewer, Print	Stream of Vector Graphics Commands
Firefox, Print, Save	Stream of Vector Graphics Commands
Firefox, Print, Print	[(Exhi)7(bi)7(t)7(A)-8(.)-20( )]
Chrome	Subset of Edge Outputs
Desktop, Save (all flows, including loaded again in Adobe)	[(Exhibi)-2(t A.)]
Desktop, Edited Save (all flows)	[(Exhibi)-2(t)-2.67(A.)-2( )]
Desktop, Edited/Unedited Print	[(E)-0.8398(xhi)-0.8320(bi)-0.8320(t)-0.8320( )-10(A)-0.1680(.)10( )]
Word 365, Mac OS	
Safari, Chrome, Firefox, Edge	Subset of Windows 10, Edge
Desktop, Save (all flows)	[(E)0.2(xhi)0.2(bi)0.2(t)0.2(A)-0.2(.)-0.104( )]
Desktop, Edited Save (all flows)	[(E)0.2(xhi)0.2(bi)0.2(t)0.2( )-0.136(A.)0.232( )]
Word 2016, Windows 10	
Save	[(Ex)-8(hibi)6(t A.)]
Save, Edited	[(Ex)-8(hibi)6(t)-2.67(A.)-2( )]
Print	[(E)-0.8398(x)-10(hi)-0.8320(bi)7(t)-0.8320( )-10(A)-0.1680(.)10( )]
Google Docs, Windows 10	
Save (All flows)	[(Exhibit A.)]
Save, Adobe, Save	[(Exhibit)55.838(A.)], [(AAA)128.417(VVV)]
Chrome/Edge, Print, Adobe PDF Save	[(0123435ÿÿÿ78ÿÿÿ)] (Actual embedded text is clobbered.)
Edge, Print, Print	Stream of Vector Graphics Commands
Firefox, Download/Print, Firefox Viewer, Print, Save	File rasterized
Firefox, Download/Print, Firefox Viewer, Print, Print	Stream of Vector Graphics Commands
Firefox, Download/Print, Firefox Viewer, Print, Adobe	Stream of Vector Graphics Commands (Adobe specific)
Google Docs, Ubuntu 21.04	
Firefox/Chrome, Save (all flows)	[(Exhibit A.)]
Firefox, Download, Firefox Viewer, Print	File rasterized
Chrome, Print/System Print	[(Exhibit A.)] (Spaces tripled)
Google Docs, Mac OS	
Chrome/Firefox/Safari, Download/Print, Save	[(Exhibit A.)]
Chrome, Print, Save as Adobe PDF	Same as Ubuntu, Chrome, System Print
Chrome, Print, Preview/Sysdiag Print, Export	[(E)0.2(hi)0.2(bi)0.2(A.)], (Spaces Tripled)
Firefox, Print, Preview, Export	[(E)0.2(hi)0.2(bi)0.2(A.)]
Firefox, Download, Firefox Viewer, Print	File rasterized
Quartz PDFContext (Apple Pages), Mac OS	
Export/Print/Preview, Save	[(E)0.2(xhi)0.2(bi)0.2(t)0.2( )55.2(A)-0.2(.)], [(AAA)129.0(VVV)]
Print, Save-as-Adobe-PDF	[(Exhibit A.)], [(AAA)128.8(VVV)]
Word 2012, Windows 10	Same as Word 2016, Windows 10
All Word Versions, Windows 8.1 and Older	Subset of Windows 10
No Disp. 600 DPI, Adobe OCR	
OCR Only, Save	[(E)-2.3(x)-2.3(h)-2.3(i)-2.3(b)-2.3(i)-2.3(t)-2.3( )*[79.75 Td - 2.3 Tc]*(A)3.3(.)3.3( )3.3()]
Edit or OCR then Edit, Save	[(E)-0.12(x)-0.12(h)-0.12(i)-0.12(b)-0.12(i)-0.12(t)-0.12( )*[71.6 Td - 0.12 Tc]*(A.)]

As a result, if a document is run through Acrobat's OCR before redaction and these sidechannels are not removed. Redactions of documents produced by Acrobat's OCR are therefore not much more secure than PDFs with an unadjusted shifting scheme.

G Microsoft Word Shifting Scheme

While other shifting schemes leak proportional width information, any accumulation of information conditioned on redacted glyphs potentially leaks information. Additional redacted information leaks occur because Microsoft Word's *Save As PDF* tool converts between two glyph coordinate representations. When users edit or create a Word document, they operate on a virtual document representation, internal to Word. This representation determines how Word displays the document in the graphical user interface (GUI). Note, however, we found the resolution of the user's screen and other environmental factors do not affect this virtual representation *insofar as* the virtual representation is used for PDF generation. Included in the virtual document's representation are a set of *internal widths* used to represent glyphs' sizes.

When Word writes a PDF file, it translates the coordinates for glyphs in this virtual representation into their respective coordinates in the resultant PDF document. The PDF file contains a separate glyph width mapping, typically matching the embedded TrueType font file. A small amount of error exists between the two representations' rendering width specifications for glyphs.

A glyph's virtual width representation can be smaller or larger than that same glyph's PDF width specification. To account for this error, Word's PDF document writer accumulates each glyph's width error from left to right across each line of text twice. The first pass modifies the internal representation of each character's width according to the difference between the current internal width and the TTF width[20] of the line's prefix up to the current position, at around a 600 DPI resolution. Note this is *not to Print to PDF* workflow, so no part of this operation is dependent on the operating system's printer driver.

In the second pass, a second left-to-right accumulation occurs, accounting for error between the widths output by the first pass ad the TTF line width, converted to text space units (typicaly 1/1000th of the font size at 72 DPI). When the accumulated error hits a three text space unit threshold, Word writes a shift to the PDF and resets the accumulator. In summary, the font metrics of the line to the left of any given threshold value effect the accumulated error, leaking redacted information via the redaction's width and non-redacted glyph shifts.[21] For example, a redacted glyph with a 10 text space unit error will overflow the accumulator and the glyph's effective PDF width will be its original advance width plus a shift between 7 and 13 units depending on the accumulator's state before overflow.

We note the line's specific character ordering affects the accumulator state as different orderings will lead to different accumulator overflow positions. For example, the accumulated error could fluctuate between postive and negative three text space units for several characters and then overflow. Both the point of overflow and the magnitude are conditioned on prior glyphs' modification of the accumulated error value. For example, in Times New Roman, the digits 0 through 9 have no error and do not contribute to the accumulator. However *a lack* of a shift value can also leak information about the redacted content. Additionally, due in part to the algorithm's first pass, smaller fonts leak more information as they have more characters per line and more threshold opportunities.

We validated Edact-Ray's models for Word versions between 2007 and 2019 using several fonts on hundreds of lines of text from Wikipedia, hundreds of manual tests, and thousands of real-world cases in Section 6. Section 6 describes Edact-Ray's process for correctly identifying Word shifting

[20] This choice of TTF is Microsoft's choice. It here refers to the TTF file embedded in the PDF after the "Save As PDF" command.

[21] Monospace fonts do not contain conversion width errors.

```
int i = 0;
int trackingAdj = 0;
do {
  int accumulatedDiff = 0;
  int lastNewAdj = 0;
  int totalWidth = 0;
  int amountAdjustedSoFar = 0;
  totalWidth += fontScaledWidths[i];
  int newAdjustment = PIXEL_W(totalWidth);
  accumulatedDiff = pixelWidths[i] - newAdjustment;
  pixelWidths[i] = amountAdjustedSoFar = lastNewAdj = newAdjustment;
  uncorrectedPixelWidths[i] = newAdjustment;
  i++;
  if (i == numChars)
    break;
  do {
    totalWidth += fontScaledWidths[i];
    if (totalWidth > WIDTH_BRKPOINT)
      break;
    int origAdjustment = pixelWidths[i];
    int newAdj = PIXEL_W(totalWidth);
    newAdj -= amountAdjustedSoFar;
    int adjustmentDifference = origAdjustment - newAdj;
    trackingAdj = adjustmentDifference - accumulatedDiff;
    if (adjustmentDifference != accumulatedDiff) {
      int v28 = trackingAdj & 1;
      int v50 = trackingAdj & 1;
      if (trackingAdj <= 0) {
        trackingAdj >>= 1;
        if (adjustmentDifference < -accumulatedDiff)
          trackingAdj += v50;
        if (-newAdj >= trackingAdj)
          trackingAdj = -newAdj;
      } else {
        trackingAdj >>= 1;
        if (accumulatedDiff < -adjustmentDifference)
          trackingAdj += v28;
        if (lastNewAdj < trackingAdj)
          trackingAdj = lastNewAdj;
      }
    }
    pixelWidths[i - 1] -= trackingAdj;
    int newTrackAdj = newAdj + trackingAdj;
    amountAdjustedSoFar = amountAdjustedSoFar + newAdj;
    accumulatedDiff = origAdjustment - (newTrackAdj);
    pixelWidths[i] = newTrackAdj;
    uncorrectedPixelWidths[i] = newTrackAdj;
    lastNewAdj = newAdj;
    i++;
  } while (i < numChars);
} while (i < numChars);
```

Fig. 5. Word WYSIWYG width adjustment method.

```
if (msword_year <= 2016) {
  int leadingSpace = 1;
  float dev = 0;
  float ttf = 0;
  for (int j = i + 1; j < vs->size(); j++) {

    if (vs->getChar(j) == ' ' && leadingSpace) {
      continue;
    } else if (leadingSpace) {
      leadingSpace = 0;
    }

    float t = textSpaceWidths2007[j] / 1000;
    double d = deviceWidths[j] / msFSize2007;

    ttf += t;
    dev += d;
    double disp = ttf - dev;
    if ((disp > 0.003 || disp < -0.003) && i != vs->size() - 1) {
      int adj = disp * 1000 + 0.5;
      vs->setShift(j, adj);
      ttf = dev = 0;
    } else {
      vs->setShift(j, 0);
    }
  }
} else {
  int leadingSpace = 1;
  float dev = 0;
  float ttf = 0;
  float t = 0;
  float d = 0;
  double disp = 0;
  for (int j = i + 1; j < vs->size(); j++) {

    if (vs->getChar(j) == ' ' && leadingSpace) {
      continue;
    } else if (leadingSpace) {
      leadingSpace = 0;
    }

    t = textSpaceWidths2019[j] / 1000;
    d = deviceWidths[j] / msFSize2019;

    ttf += t;
    dev += d;
    disp = ttf - dev;

    if (((disp > 0.003) || (disp < -0.003)) && i != vs->size() - 1) {
      int adj = disp * 1000 + 0.5;
      vs->setShift(j, adj);
      ttf = dev = 0;
    } else {
      vs->setShift(j, 0);
    }
  }
}
```

Fig. 6. Adjustment routine for Word's displacment scheme

```
std::vector<double> Office::computeEditAdjustments(VectorString *vs) {
    std::vector<double> editAdjustments;
    int totalDots = 0;
    double totalDevWidth = 0;
    for (int i = 0; i < numChars; i++) {
        totalDots += (i == numChars - 1) ? uncorrectedPixelWidths[i] :
            pixelWidths[i];
        totalDevWidth += (textSpaceWidths2019[i] - vs->getShift(i)) * (
            msFSize2019 / 1000);
        double realWidth, editedWidth, adj;
        realWidth = totalDevWidth + dotsToPdfUnits2019(600);
        editedWidth = roundToDigits(dotsToPdfUnits2019(totalDots + 600), 5);
        adj = (realWidth - editedWidth) / (msFSize2019 / 1000);
        editAdjustments.push_back(adj);
    }
    return editAdjustments;
}
std::vector<double> Office::edit(VectorString *vs, std::vector<double> eAdjs,
    int i) {
    double disp;
    disp = roundf(eAdjs[i] * (msFSize2019 / 1000) * 1000) / 1000 / (msFSize2019
         / 1000);
    vs->addShift(i, disp);
    adjust(vs, i);
    return computeEditAdjustments(vs);
}
void Office::editSuffix(std::vector<double> savedEAdjs, VectorString savedVs,
    VectorString *suf) {
    double guessShift;
    double checkShift;
    double adjShift;
    bool matches = true;
    for (int i = 0;i < suf->size()-1;i++){
        int ind = savedVs.size() - suf->size() + i;
        guessShift = roundf(10 * savedVs.getShift(ind));
        checkShift = roundf(10 * suf->getShift(i));
        if (ind == savedVs.size() - 1) {
            savedVs.setShift(ind, suf->getShift(i));
            break;
        }
        adjShift = roundf(10*savedEAdjs[ind]);
        if (guessShift + adjShift == checkShift) {
            savedEAdjs = edit(&savedVs, savedEAdjs, ind);
        } }
    for (int i = 0;i < savedVs.size()-1;i++){
        if (savedVs.getChar(i) == L'-') {
            savedEAdjs = edit(&savedVs, savedEAdjs, i - 1);
            savedEAdjs = edit(&savedVs, savedEAdjs, i);
        } } }
```

Fig. 7. Microsoft Word edit history shifting scheme algorithm.

```
for (int i = 0; i < numChars; i++) {
  fontScaledWidth = widths[i] * fontSize * 2;
  fontScaledWidths.push_back(fontScaledWidth);
  if (msword_year == 2007) {
    float textSpaceW =
        roundf(roundf(((float) widths[i]) / UNIT_TEXT_SPACE_2007 * 10000) / 10);
    textSpaceWidths2007.push_back(textSpaceW);
  } else {
    double textSpaceW = roundf(((double) widths[i]) / UNIT_TEXT_SPACE_2019 *
        1000.0);
    textSpaceWidths2019.push_back(textSpaceW);
  }
  uncorrectedPixelWidths.push_back(0);
}
if (justif) {
  justify();
}
computeWYSIWYG();
```

Fig. 8. Word initialization of widths.

schemes. We also note that Microsoft Word's shifting scheme metadata depends on the Word document's edit history. Appendix Figure 7 gives code for edit history modeling.

G.0.1 Word Glyph Positioning Algorithms The first routine (Figure 8) intializes a set of widths from two files, widths corresponds to TTF widths, and pixelWs are Word's internal widths for each character. It then initializes the WYSIWYG sizes. The next routine (Figure 5) handles calculating the WYSIWYG widths for each character by accumulating errors between the two width representations, consisting of a tested loop that checks sets of runs of characters: Then (Figure 6), once an actual line of text needs to be adjusted, the deviceWidths initialized during the WYSIWYG modification are checked for overflow. The types of floating point variables, e.g. float vs. double, are precisely chosen and must be identical to the original algorithm or the result will be incorrect.

H Latex, LibreOffice, Other PDF Producers

LibreOffice LibreOffice [14] includes a PDF writer as part of its Visual Class Library (VCL), and, at the time of writing, embeds shifts in the drawHorizontalGlyphs method of the PDFWriterImpl class. This method appears to be similar to Microsoft Word in function: shifts are applied whenever the native advance width of the PDF glyph does not match the Pixel x,y coordinates that LibreOffice uses internally, derived from the SalLayout (layout engine) class. However, LibreOffice does not use a second width map for the characters in a given font, and therefore likely leaks less redacted information than Microsoft Office's shifting scheme.

LaTeX (TeX Live) The determination of shifts inside LaTeX is dependent upon the particular flow that is used [17]. For the pdfTeX flow, interglyph shifts are decided by the pdf_begin_string procedure. The LaTeX glyph placement algorithm is quite complex, however, our analysis did find one section of the code for determining shifts appears to round the difference between the current horizontal glyph render position and the start of the current text object. Depending on the precision of the rounding, this could leak additional information about the order of glyphs used. Other flows, such as XeLaTeX, can optionally include more complex calculations that determine how to shrink and expand glyphs to fit a certain amount of space and may leak large amounts of redacted information. However, LaTeX redactions are uncommon.

Other Producers We found other producers, such as iText, where the default is unadjusted text objects, are accounted for by the schemes presented in the main body of this paper (Section 2).

H.1 Document Scanners

Many document scanners also come with OCR features. We examined the output of three different document scanners: a Canon MP Navigator EX, Xerox AltaLink C8145, and HP MFP M227fdn. Of the three, the Canon and Xerox machines provided built-in software for document OCR. We examined the Canon's glyph shifting scheme first, and found redactions on PDFs produced by this workflow to be equally as vulnerable to deredaction as unadjusted schemes. The Xerox machine was not as simple: the Xerox machine's OCR text objects are positioned stochastically without space characters in between them. After testing several redaction tools on the Xerox document, we found it was as vulnerable as a document with an unadjusted scheme as long as redaction was performed by removing glyphs rather than entire text objects.