A Multi-leveled Approach for Detection of Coercive Malicious Documents Employing Optical Character Recognition

Josiah Smith

A Multi-leveled Approach for Detection of Coercive Malicious Documents Employing Optical Character Recognition

GIAC (GCIH) Gold Certification

Author: Josiah Smith, josiahraysmith@gmail.com
Advisor: Lenny Zeltser

Accepted: February 12, 2021

Abstract

Authors of malicious documents often include a graphical asset used to lure the potential victim to “enable editing” and to “enable content” to activate the macro’s embedded logic. While these graphical lures vary in theme, language, and content, they commonly have similar coercive text. Using Optical Character Recognition to produce text files of the images provides the ability to anchor the images’ contents. While attackers have been known to intentionally manipulate images to bypass OCR-based detection, some additional techniques can surface the textual contents. Optical Character Recognition can be utilized to track, pivot, and cluster malicious campaigns, identify new TTPs, and possibly provide attribution against adversaries.

1. Introduction

There is no question that document-based malware is a popular attack vector that puts an organization's posture at risk. *A Multi-leveled Approach for Detection of Coercive Malicious Documents Employing Optical Character Recognition (OCR)* details the attempts to evade OCR-based detection by manipulating embedded graphical assets while maintaining the subtlety of the lure.

OCR is a technology used to identify text within images. OCR-based detection is complementary to other detection methodologies used by security operations teams. Based on research of modern attack campaigns, multiple threat actors use these malware lures to convince users to enable active content. Detection at this point can potentially break the kill chain and prevent the next organizational data breach.

This testing methodology in this research is defined by utilizing a constant set of graphical assets to measure the effectiveness of two different OCR products on the images. The resulting text is processed with a Yara rule designed to detect coercive language instructing users to activate the embedded logic. The images will then be modified through multiple techniques ranging from transparency, colors, blurriness, and font types, and then processed through the OCR engines and evaluated again. In addition to the pre- and post-manipulation results, some alternative detection techniques are assessed for use in the detection chain.

While some images are intentionally manipulated images to bypass OCR-based detection, additional techniques can surface the textual contents. Optical Character Recognition can be utilized to track, pivot, and cluster malicious campaigns, identify new TTPs, and possibly provide attribution against adversaries.

2. Optical Character Recognition

Optical Character Recognition has become an important and widely used technology. Among its many practical applications are money changing machines, mobile device check deposit, office scanners, and automation within the postal and shipping industry. Although the technology has been a topic of interest and research for many years, developing OCR with capabilities comparable to human interpretation is a sustainable challenge. OCR technology

presents a complex problem set due to the variety of languages, fonts, and styles in which text can be written. Other nuances ranging from image quality to intentional tampering can cause inconsistent results throughout the process.

The process of OCR is a complex activity that can be described as a sequence of phases. As described by Islam, Islam, and Noor (2017), the steps are as follows:

- Image acquisition: The process of acquiring the image.
- Pre-processing: Enhances the quality of the image
- Character segmentation: Separates the image into component characters
- Feature Extraction: Extracts other features form the image
- Classification: Classifying characters into their appropriate category
- Post-processing: Improve accuracy of results

While there are many intricacies and various approaches across different OCR technologies, this research's scope will cater to the application of detecting coercive lures for malicious documents. The two specific OCR engines that will be described and utilized to derive text from the graphical lures are Tesseract and iDRS.

Tesseract is an open-source text recognition engine that is freely available under the Apache 2.0 license (Tesseract, 2020). The tool can be used directly from the command line, through its full-featured API, or various third-party contributed graphical user interfaces. Additionally, there are dozens of languages supported throughout its versions.

The installation process for Tesseract is relatively straightforward and contains two portions to include the engine itself and the training data for the selected language. The package is directly available for many Linux distributions. Simply running the following commands will install on Ubuntu systems:

```
# apt install tesseract-ocr
# apt install libtesseract-dev
```

The second OCR engine used to extract text from malware graphical lures is iDRS from IRIS. The iDRS OCR toolkit is a commercial application that offers the capability to convert all images into indexed and editable files. Considering the installation is a bit more extensive, and the installation media is not freely available since it is a licensed application, the installation steps are not detailed.

It is important to note that while there is an opportunity to compare the two OCR engines that will be utilized as tools to extract the textual content from graphical lures, this research intends not to compare the two products. Instead, they will be used as standard toolsets for the experimentation of detecting malware lures utilizing OCR technologies. Additionally, there are many different pre-processing features of both engines that could be tuned to achieve higher efficacy of text extraction and identification. When considering each toolset's extensiveness and consistent effects, the default configurations will be used and will not be tuned throughout the process.

2.1. Malware Graphical Lures

After installing the OCR engines, the next step in the research methodology is to acquire samples of coercive graphical lures used in malicious documents. There are many ways to address this step which range from an automated to a manual approach. One solution to acquire the coercive lure without opening the weaponized document is downloading the image from some repositories or websites focused on this topic. A particular relevant source of these malware lures is Dr. Josh Stroschein's maldoc template GitHub repository (Stroschein, 2021). Another thoroughly developed source for images to perform OCR on is the Malware Lures Gallery from InQuest (InQuest, 2020).

If the image's extraction is sourced directly from a document, there are a few different tools and techniques to extract them, but the effort and effectiveness are dependent on that specific file type. In the case of .docx and .docm files, those are actually disguised zip files, and the images are found within the media directory of the unzipped contents. The following bash command will loop through every .docx file and extract the images to their respective and newly-created directory.

```
for L in *.doc*; do mkdir -p "$L"-images && unzip "$L" "word/media*" -d "$L"-images; done
```

The process to extract the images is very similar for .xlsx files. The differing aspect is the images are stored in the "xl/media/" portion of the unzipped document. The following script can automate the extraction of images from multiple files.

```
#!/bin/bash
for L in *
do
    mkdir -p "$L"-images
    unzip "$L" "xl/media/*" -d "$L"-images
done
```

While the image extraction from the aforementioned file types proved trivial, the extraction from other file types like .doc can be a bit more challenging due to the streams associated with the Object Linking and Embedding (OLE) file format. Ogden, Roberts, and Sayre (2018) describe OLE, "a complex binary format that is not easily manipulated", but represent the compressed archives formats' flexibility for manipulating the contents of the file.

Another favorite tool to assist in carving images out of the documents is Foremost. Originally developed by the United States Air Force Office of Special Investigations and the Naval Post-Graduate School Center for Cybersecurity and Cyber operations, Foremost is a console program to recover files based on their data structures. One practical use-case shown by Smith (2020) is the extraction of executables embedded within images. However, this use case will show the carving images from documents. Looking at the malicious document b93c1b5898ee3d02d1f7996c90256099 (VirusTotal, 2021) masquerading as Ginny Hergott's resume, Foremost carves out one image.

```
$ foremost b93c1b5898ee3d02d1f7996c90256099 && md5sum output/{png,jpg}/* 2>
/dev/null

0e822212fa479958692131134e28de53  output/png/00000008.png
```

Figure 1 shows the rendering of the extracted .png.

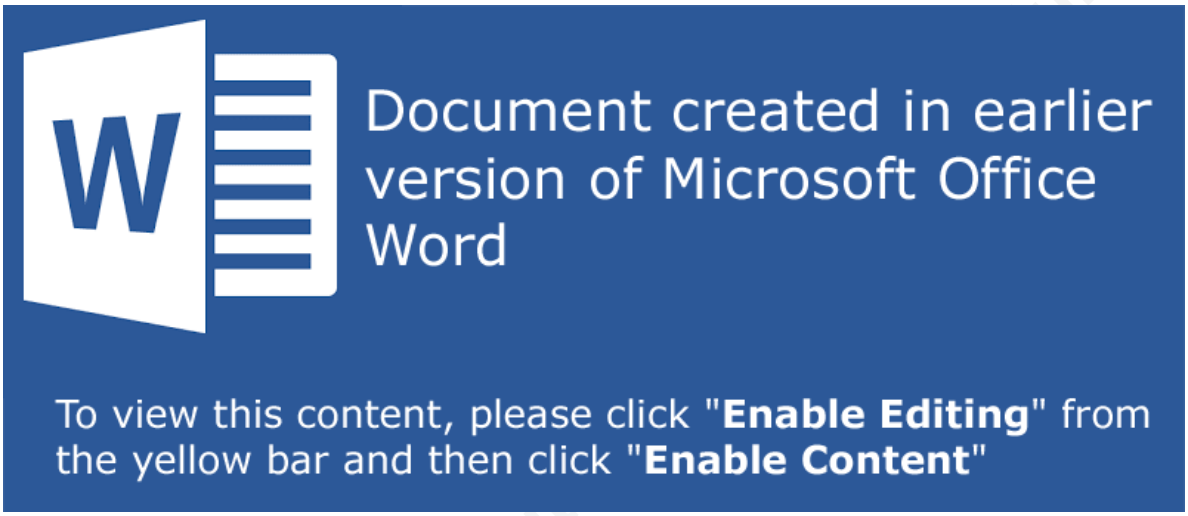


Figure 1: 0e822212fa479958692131134e28de53 (cropped for sizing)

Another approach to carving out the images found within OLE files is Didier Stevens's oledump.py (Stevens, 2021). However, this technique is generally more difficult as the stream containing the image is unknown at the beginning of the analysis, and on occasion, there is some data or other chaff bytes ahead of the image's magic. Demonstrating with the file md5: aaa33b12b551840719b1df0d6a8ac731 (VirusTotal,2021), it is apparent that some bytes need to be removed before the image can be rewritten. Furthermore, while earlier analysis found the data in stream 5 of 14, it is necessary to identify the correct stream.

```
oledump.py -a -s 5 aaa33b12b551840719b1df0d6a8ac731 | more
00000000: 9E 3F 00 00 44 00 64 00  00 00 00 00 00 00 08 00  .?..D.d.........
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 51 26 C0 11  ............Q&..
00000020: DB 03 DB 03 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040: 00 00 00 00 0F 00 04 F0  4C 00 00 00 B2 04 0A F0  ........L.......
00000050: 08 00 00 00 01 04 00 00  00 0A 00 00 43 00 0B F0  ............C...
00000060: 28 00 00 00 04 41 01 00  00 00 05 C1 10 00 00 00  (....A..........
00000070: 06 01 02 00 00 00 FF 01  00 00 08 00 36 00 37 00  ............6.7.
00000080: 31 00 32 00 38 00 30 00  32 00 00 00 00 00 10 F0  1.2.8.0.2.......
00000090: 04 00 00 00 00 00 00 80  62 00 07 F0 FE 3E 00 00  ........b....>..
000000A0: 06 06 8B 86 8A 37 D9 75  0B 03 08 A5 2B DC C1 4A  .....7.u....+..J
000000B0: 59 92 FF 00 DA 3E 00 00  01 00 00 00 44 00 00 00  Y....>......D...
000000C0: 00 00 7C 00 00 6E 1E F0  D2 3E 00 00 8B 86 8A 37  ..|..n...>.....7
000000D0: D9 75 0B 03 08 A5 2B DC  C1 4A 59 92 FF 89 50 4E  .u....+..JY...PN
000000E0: 47 0D 0A 1A 0A 00 00 00  0D 49 48 44 52 00 00 02  G........IHDR...
000000F0: 8E 00 00 01 2F 08 06 00  00 00 CB A6 40 36 00 00  ..../.......@6..
```

Josiah Smith, josiahraysmith@gmail.com

Overcoming this uncertainty is simple enough with some rudimentary scripting skills. The following Python script will match, carve, and write the embedded PNG image back into the current working directory.

```
#!/usr/bin/env python3
import re
import sys
with open(sys.argv[1], "rb") as fp:
    contents = fp.read()
    match = re.search(b"\x89\x50\x4e\x47\x0d\x0a\x1a\x0a", contents)
    if not match:
        sys.exit("No match")
    with open("carved_image", "wb") as carved_image:
        carved_image.write(contents[match.start():])
        sys.exit("PNG Image Found")
```

There is still a problem identifying which stream to look for the image data, but a simple BASH loop here will write out all the streams to be looped through the PNG image finder.

```
$ for L in `oledump.py -d aaa33b12b551840719b1df0d6a8ac731 | cut -d: -f1`; do
oledump.py aaa33b12b551840719b1df0d6a8ac731 -d -s$L > stream$L ; done

$ ls stream*| while read L; do ./pngfind.py $L; done
No match
No match
No match
No match
No match
No match
No match
No match
No match
PNG Image Found
No match
No match
No match
No match
$file carved_image && md5sum carved_image

carved_image: PNG image data, 654 x 303, 8-bit/color RGBA, non-interlaced
7490be961ba7e861590271c6e033f431  carved_image
```

The resulting image shown in figure 2 was found in the stream and carved with oledump.py after removing the chaff bytes at the head of the data stream.

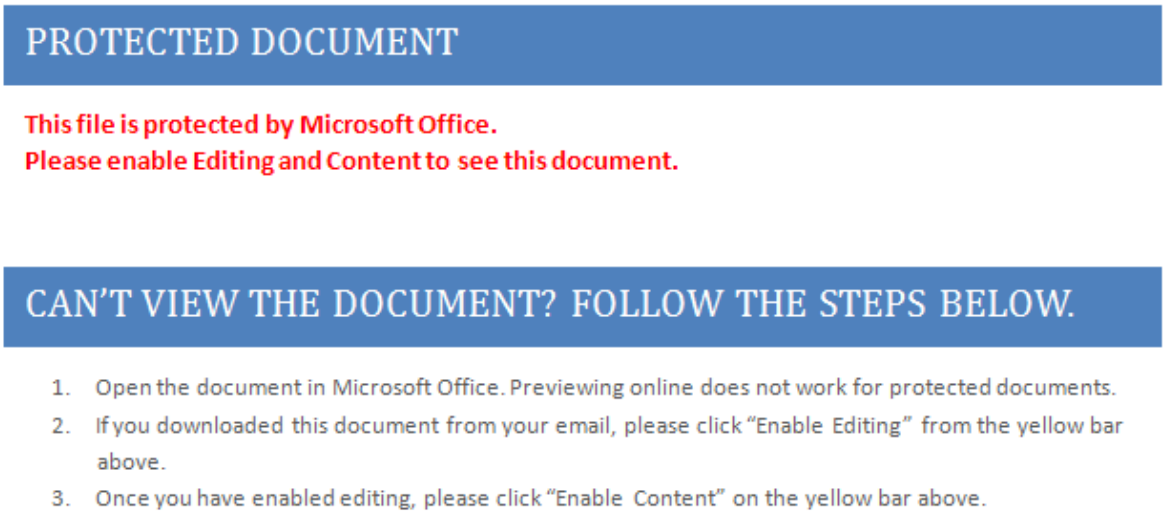


Figure 2: 7490be961ba7e861590271c6e033f431

2.2 YARA Detection Logic

Now that multiple approaches have been described for the acquisition of images from documents, the next step is to perform OCR on the images with the two engines and acquire the first pass results. The variable measured is the output of the OCR tools from multiple controlled images and two different OCR products. After impacting the detection from manipulating images, the detection logic is modified to detect the images. The following YARA rule can detect malware lure coercion and may be improved to cover the difference in output.

```
rule OCR_Rule
{

        strings:

        $enable1 = "Enable Content" nocase ascii wide
        $enable2 = "Enable Editing" nocase ascii wide
        condition:
                        any of ($enable*) and filesize < 1KB
}
```

3. Pre-manipulation

3.1. Image 1 Pre-manipulation

The first image analyzed with OCR and the YARA based detection rule will be the following MS Office Word-themed Lure originally derived from file md5: 138e3164dd69df539809a0bc5cd37d36. As previously discussed, this Microsoft Word 2007+ file can have the image extracted by unzipping the file and searching in the media directory.

```
$ unzip 138e3164dd69df539809a0bc5cd37d36 "word/media*" -d images
Archive:  138e3164dd69df539809a0bc5cd37d36
 extracting: images/word/media/image1.png
```

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

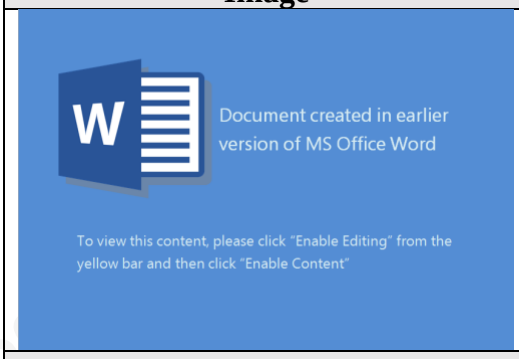
Image	Tesseract Output	iDRS Output
	W Document created in earlier version of MS Office Word To view this content, please click "Enable Editing" from the yellow bar and then click "Enable Content"	Document created in earlier version of MS Office Word To view this content, please click "Enable Editing" from the yellow bar and then click "Enable Content"
YARA Signature Hits	$ yara OCR.rul image1.png.ocr -s OCR_rul image1.png.ocr 0x96:$enable1: Enable Content 0x5f:$enable2: Enable Editing	$ yara OCR.rul image1.png.txt -s OCR_rul image1.png.txt 0x93:$enable1: Enable Content 0x5f:$enable2: Enable Editing
Coercive Detection Success	Yes	Yes

Figure 3: Image 1 Summary

3.2. Image 2 Pre-manipulation

The second image that will be analyzed with OCR and the YARA-based detection rule will be the following MS Office 365 originally derived from file md5: 12453bda42e40501077b1d202a338648.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the template.

Image	Tesseract Output	iDRS Output
	‘mOffice 365 You are attempting to open a file that was created in an earlier version of Microsoft Office. lfthe file opens in Protected View, click Enable Editing, and then click Enable Content	n Office365 You are attempting to open a file that was created in an earlier version of Microsoft Office. If the file opens in Protected View, click Enable Editing, and then click Enable Content
YARA Signature Hits	$ yara ../OCR.rul image1.jpg.ocr -s OCR_Rule image1.jpg.ocr 0xbb:$enable1: Enable Content 0x9c:$enable2: Enable Editing	$ yara ../OCR.rul image1.jpg.txt -s OCR_Rule image1.jpg.txt 0xbd:$enable1: Enable Content 0x9e:$enable2: Enable Editing
Coercive Detection Success	Yes	Yes

Figure 4: Image 2 Summary

3.3. Image 3 Pre-manipulation

The third image analyzed with OCR and the YARA based detection rule was the following Amazon gift card-themed Lure originally derived from file md5: 678aff9ae66a0fb94043261eeeecaec5. As previously described, the embedded image is extracted with oledump.py and pngfind.py.

```
$ oledump.py 678aff9ae66a0fb94043261eeeecaec5 -d -s5 > stream5
$ ./pngfind.py stream5
$ file output
output: PNG image data, 1017 x 1092, 8-bit/color RGBA, non-interlaced
```

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

Image	Tesseract Output	iDRS output
	amazon I] Office This document created with MicroSOft Office newest version. Please,.press “Enable Content’I to resolve.	$500 amazon • Ice This document created with Microsoft Office newest version. Please, press “Enable Content” to resolve.
YARA Signature Hits	$ yara ../OCR.rul image1.png.ocr -s OCR_Rule image1.png.ocr 0x5f:$enable1: Enable Content	$ yara ../OCR.rul image1.png.txt -s OCR_Rule image1.png.txt 0x6c:$enable1: Enable Content
Coercive Detection Success	Yes	Yes

Figure 5: Image 3 Summary

3.4. Image 4 Pre manipulation

The fourth image analyzed with OCR and the YARA based detection rule was the following Happy New Year - Apple Store Gift Voucher Lure originally derived from file md5: 5e870ca9e50adec114a98eab76f5b49c.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

Image	Tesseract Output	iDRS Output
	- d AppleStore ““Gift Voucher figggy New Year MWH§MWX§§L I ‘ _. ._ Office This document created with Microsoft Office newest version; ' ' ' ' ' ' ” Please, press “Enable Content” to resolve. ‘Fit-hum’ Adam Information Invoice Number Iwoice Dale terms Ship Ode “13003 mm Tfisorderisswiectbflwle'sSdsadRemPoicies. . w1m.we.cWWies.hm .	9 Apple Store Gift Voucher You make us proud every year. • ice This document created with Microsoft Office newest version. Please, press “Enable Content” to resolve. For a total of $25.50 Addltlonal Information lnvoce Nm DPf Invoice Date I l!UllS Shup Date 9299613083 Dec, 2020 Credit Card 1 his order is suti,ect to Apple's Sales and Refunds Policies hl1p //store apple com/CatalogflJ S/l1P;iges.1s;itespo11c1es html
YARA Signature Hits	$ yara ../OCR.rul image1.png.ocr -s OCR_Rule image1.png.ocr 0xbb:$enable1: Enable Content	$ yara ../OCR.rul image1.png.txt -s OCR_Rule image1.png.txt 0x99:$enable1: Enable Content
Coercive Detection Success	Yes	Yes

Figure 6: Image 4 Summary

3.5. Image 5 Pre-manipulation

The fifth image analyzed with OCR and the YARA based detection rule was the following MS Office Word-themed Lure originally derived from file md5: b1d4a944e6a71d17037664135b3498eb. Unfortunately, detection capability was foiled by introducing Spanish text within the coercive lure.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure. While the OCR engines appeared to efficiently carve out the text of “Abilita modifiche” and “Abilita contenuto”, the YARA rule did not include those strings. This situation is an excellent example of identifying False Negatives “in the wild” and using them to improve the efficacy of an organizations signature set.

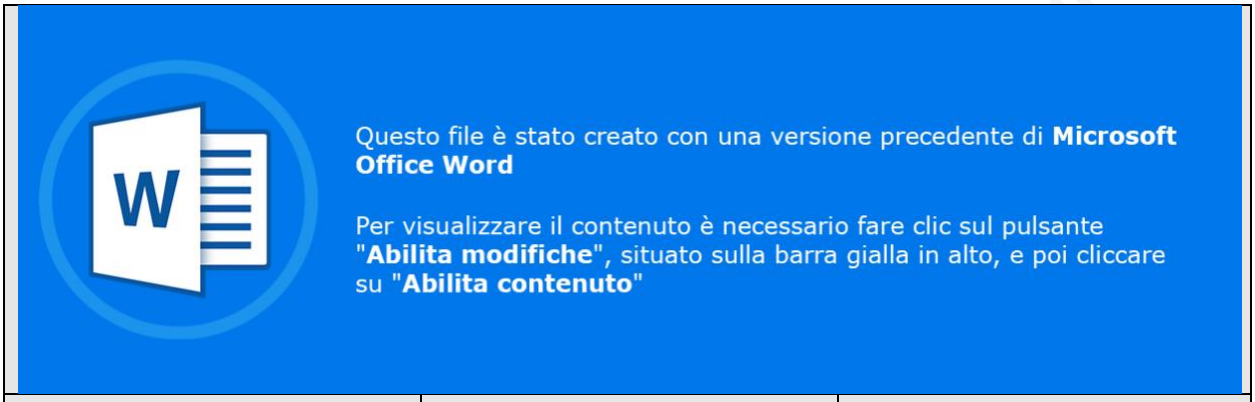


Image	Tesseract Output	iDRS output
	Questo file (‘9 state creato con una versione precedente di Microsoft Office Word Per visualizzare i| contenuto é necessario fare clic sul pulsante ”Abilita modifiche", situato sulla barra gialla in alto, e poi cliccare su "Abilita contenuto"	Questo file e stato creato con una versione precedente di Microsoft Office Word Per visualizzare ii contenuto e necessario fare clic sul pulsante "Abilita modifiche", situate sulla barra gialla in alto, e poi cliccare su "Abilita contenuto"
YARA Signature Hits	No hits	No hits
Coercive Detection Success	FAIL	FAIL

Figure 7: Image 5 Summary

4. Attempts to Evade OCR Detection by Manipulating Images

The following section details the attempts to evade the detection produced by OCR text extraction and the provided YARA rule. While the evasion would be easily accomplished by completely removing the suggestive instructions to enable the embedded logic, the spirit of the coercive image is designed to stay the same while subtly editing the image to maintain the apparent legitimacy and aspect of social engineering. Additionally, the results are described after the first attempt and not continuously tuned until the evasion is successful. Much like malware authors who test their attacks to gauge success against antivirus solutions before the campaign begins, multiple iterations are likely to be more successful.

4.1. Image 1 Post-manipulation

The first attempt at manipulating an image to evade the previously successful OCR derivative string detection is performed on the following image. For this example, the technique used was to change the strings' font color, providing instruction for enabling the document's macros. The color was changed to a similar color as the image background while maintaining the text's readability. The technique was successful against the first OCR engine.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

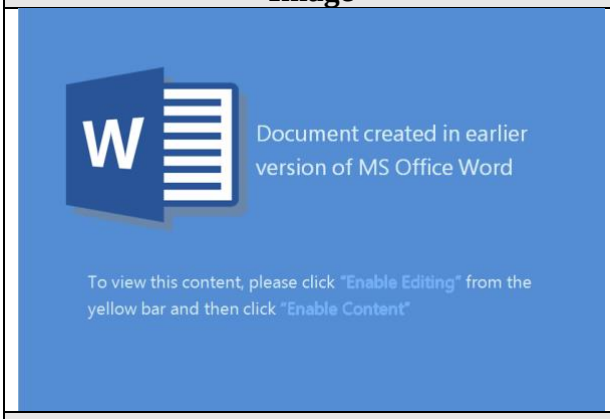
Image	Tesseract Output	iDRS Output
	W Document created in earlier version of MS Office Word To view this content, please click from the yellow bar and then click	Document created in earlier version of MS Office Word To view this content, please click "Enable Editing" from the yellow bar and then click "Enable Content"
YARA Signature Hits	None	$ yara ../../OCR.rul Image1-manipulated.png.txt -s OCR_Rule Image1-manipulated.png.txt 0x93:$enable1: Enable Content 0x5f:$enable2: Enable Editing
Coercive Detection Success	No	Yes

Figure 7: Altered Image 1 Summary

4.2. Image 2 Post-manipulation

For the second image, a yellow box was drawn around the instructive strings to accentuate them to the unbeknownst user. Additionally, a lighter vertical line was drawn between the words to inject another character into the output. The effects disrupted the Tesseract engine output accuracy and eliminated half of the detected strings from iDRS.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

Image	Tesseract Output	iDRS Output
	‘mOffice 365 You are attempting to open a file that was created in an earlier version of Microsoft Office. lfthe file opens in Protected View, click EnableIEditing, and then click EnablelContent	Office365 You are attempting to open a file that was created in an earlier version of Microsoft Office. If the file opens in Protected View, click Enable Editing, and then click Enable/Content
YARA Signature Hits	None	$ yara ../../OCR.rul image2-manipulated.jpg.txt -s OCR_Rule image2-manipulated.jpg.txt 0x9a:$enable2: Enable Editing
Coercive Detection Success	No	Yes. 1 of two strings

Figure 8: Altered Image 2 Summary

4.3. Image 3 Post-manipulation

The third image that was manipulated to avoid detection is the Amazon Gift Card-themed image. For this example, the image was opened in the standard Microsoft Paint Application, and the text was replaced with a different text box containing a zero ASCII character instead of the “o” in the word “content.” Additionally, a thin eraser was used against the text to lower the letters’ “crispness.”

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure. While the extracted strings were close, this manipulation effort was able to bypass detection based on both engines' output.

Image	Tesseract Output	iDRS Output
	amazon fl Office This document created with MicrosOft Office newest version. Please,.press "Erilabie COntent" to resolve.	$500 amazon --· • ice This document created with Microsoft Office newest version. Please, press "Enbie COntent" to resolve.
YARA Signature Hits	None	None
Coercive Detection Success	NO	NO

Figure 9: Altered Image 3 Summary

4.4. Image 4 Post-manipulation

The Apple Gift Store image was manipulated with a blur filter over the "Enable Content" string. The filter swapped the text and background color to implement a black box around the white text. This specific technique disrupted the accuracy of the IDRS engine, but Tesseract still provided effective translation for the subsequent YARA inspection.

The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

Image	Tesseract Output	iDRS Output
	- 6 AppleStone “Gift Voucher baggy New Year mmfimew _. ._ Office This document created with Microsoft. Office newest version; ' ' ' ' ' ' ” Please, press” Enable Content” to resolve. ‘rm us a m’ Addltloml Intonation Invoice Nunbe: hvoice Dale Tums Ship Ode 9299813003 the m Cad m, m TNsoMefissnfiectbApple’sSdaaflRelt-Idsm. . wyme.we.cwwsnmuhm .	9 Apple Store Gift Voucher You make us proud every year. • ice This document created with Microsoft Office newest version. Please, press "Ena.(,f e Cont:ent:" to resolve. For a total of $25.50 Addltlonal Information lnvoce Nm DPf Invoice Date I l!UllS Shup Date 9299613083 Dec, 2020 Credit Card 1 his order is suti,ect to Apple’s Sales and Refunds Policies hl1p //store apple com/CatalogflJ S/l1P;iges.1s;itespo11c1es html
YARA Signature Hits	$ yara ../../OCR.rul image4-manipulated.png.ocr -s OCR_Rule image4-manipulated.png.ocr 0xb1:$enable1: Enable Content	None
Coercive Detection Success	Yes	No

Figure 10: Altered Image 4 Summary

4.5. Image 5 Post-manipulation

The Spanish language-themed Microsoft word image’s initial results were found to have exceptional extraction from both OCR engines. However, the multi-language support was not included in the YARA rule used for detection. For this post-manipulation analysis, the individual strings were added into the rule to support the Spanish strings. The manipulation of the image was done by blurring all of the text. While a bit more aggressive, the text is still readable, and this technique has been seen “in the wild”, proving the technique is successful in both aspects.

Neither OCR engines’ derived text was even close to accurate and ultimately evaded detection from both engines. The summary of the results below describes efficient OCR extraction and YARA detection from both OCR engines on the graphical lure.

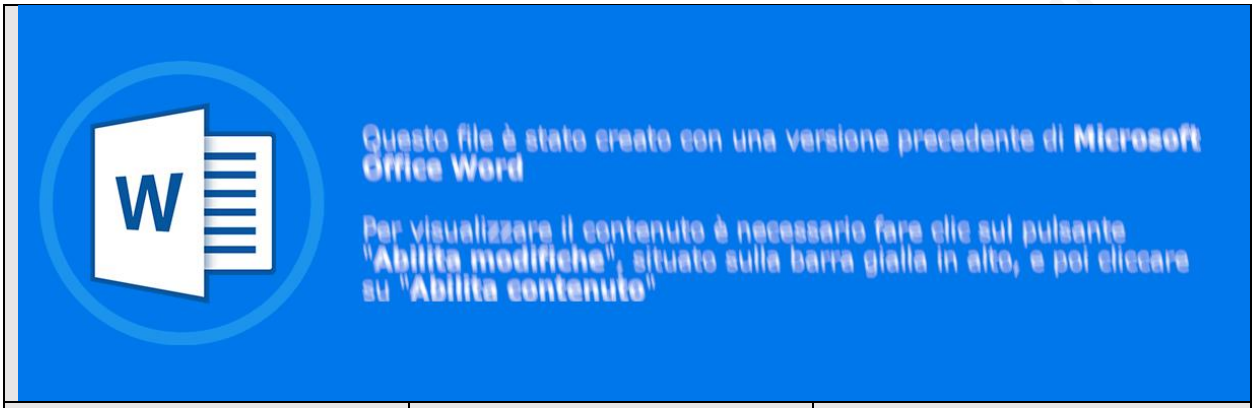


Image	Tesseract Output	iDRS output
	\“HHMH NH 9* MMH \ H‘MH \\‘H mm \M MHHH Wm MWMH \H M'HHIH“ Um”! Wm” “M Human Mn N mnhumm fl mu wmmm Mw ~ |h um .mhnmw “A“..“N HHHN". “0|” HHHM‘H HHH!‘ "NHM \H!‘..!\ H‘ MHH H .‘H‘ \ H\ \ vHH nu “Amum umhnuhs”	~UG&tl f\11 i &tttl \ll1Gltl \1\lll Ullt\ \Jll111tll1Q p111n1tlo1,t\\ till ~,1cro1oft Gff loo wortl PG11 \ JIIUftllllt\1’ \) II \1\llltlllUtO G llGQQiit\111\l f~I’ \\ ollo illl ~Ulillltl Abllita modittehe” etueto »ulla barre gills In alto, e pol eleeare au 11Allllllm oontonum11
YARA Signature Hits	None	None
Coercive Detection Success	NO	NO

Figure 11: Altered Image 5 Summary

5. Evasion Detection with DHASH

The previous image manipulations have proven the capacity of evading OCR-based detection against a standard set of OCR engines and a YARA rule. It’s responsible to acknowledge some mitigative factors to still contribute detection on the altered images. One such technique that can be used is perception hashing. This difference hash can be useful for detecting duplicate images and the slightly modified renditions used throughout the research. The following algorithm is known as DHASH, based on Neal Krawetz’s study (2013). The algorithm works by:

- Converting image to grayscale
- Reducing to a 9x9 thumbnail
- Producing a 64-bit row hash and 64-bit column hash
- Combining the two values

The installation and use for DHASH is relatively trivial. Analysts can run:

```
$ pip install dhash
$ python -m dhash [filename.png]
```

The following results depict the similarities between the images derived from the original maldoc and the edited graphics. Note the similarity between the perception hash despite the differing cryptographic MD5 hash. The threshold for defining similar based images based on perception hashing is subjective based on the implementation. For this use-case, the difference of 10% or less will be defined as a similar or manipulated image. With respect to the utilization of DHASH-based threat hunting or security monitoring, the images from malicious documents will need to be acquired, analyzed, and curated into a derived detection rule to be effective.

5.1. Image 1 Perception Hashing

```
$ md5sum image1*
931ac00fcbf461fae4fd3ee9f1f8ff02  image1-manipulated.png
d5fb7eb635c3c6fe0b2cfc4118627f08  image1.png

$ python -m dhash image1-manipulated.png
9069484c69b0c081107f7f80cf7f0003
$ python -m dhash image1.png
9069484c69b4c081107f7f80cf7f0003

$ python -m dhash image1.png image1-manipulated.png
1 bit differs out of 128 (0.8%)
```

5.2. Image 2 Perception Hashing

```
$ md5sum image2*
2602785981f896ab4371f40690370ac7  image2.jpg
4b8326ba41996cb41d36b6c0fce699c0  image2-manipulated.jpg

$ python -m dhash image2.jpg
202b2b230049512d80607fff000d40ff
$ python -m dhash image2-manipulated.jpg
202b2b230049532d80607fff000d40ff

$ python -m dhash image2.jpg  image2-manipulated.jpg
1 bit differs out of 128 (0.8%)
```

5.3. Image 3 Perception Hashing

```
$ md5sum image3*
95955cd4de263fc282fadd832e300b17  image3-mapipulated.png
6307f999679ef3f53f73d26a5590c334  image3.png

$ python -m dhash image3.png
083571193e5b4d82000cffc13cfeff82
$ python -m dhash image3-mapipulated.png
083571193e7b7982000cffc13cfeff8e

$ python -m dhash image3.png image3-mapipulated.png
6 bits differ out of 128 (4.7%)
```

5.4. Image 4 Perception Hashing

```
$ md5sum image4*
55d559abd3a9ab3c98633681bb84e4be  image4-manipulated.png
be400144480ff3d883221711914fce1d  image4.png

$ python -m dhash image4.png
302c0f3b1f1f2b376040007f411c3f63
$ python -m dhash image4-manipulated.png
302c0f3b1b0f0f336040007f41103f63

$ python -m dhash image4-manipulated.png image4.png
7 bits differ out of 128 (5.5%)
```

5.5. Image 5 Perception Hashing

```
$ md5sum image5*
211b9216186b9edeb8e5d70ef33bf19d  image5-manipulated.png
41ad7749103edc1fa5f1232cba5e51d2  image5.png

$ python -m dhash image5.png
0281a2ae88a0a49040fff700ff0000bf
$ python -m dhash image5-manipulated.png
2481a6ae88a0a49500fff700ff0000bf

$ python -m dhash image5.png image5-manipulated.png
7 bits differ out of 128 (5.5%)
```

6. Alternative Approach for Detecting Images (Halogen)

Throughout the research, a handful of alternative tools proved interesting in similarity for defensive purposes. An alternative approach for detecting files that have been embedded within malicious documents is the command-line tool Halogen. The tool automates the creation of YARA rules against image files embedded within documents (Eaton & Roersma, 2020).

This technique can provide additional coverage for maldoc lures that are not detected with OCR. Similar to Amini and Remen's (2019) method to track and detect malicious documents while anchoring the XMP IDs, the previous examination of a document containing the graphical asset must be sourced to develop the detection logic. The tool is user friendly; after installation and a look at the help menu, the Yara rule is generated with the five maldoc samples used for the initial acquisition of the graphical lures.

```
python3 halogen.py -h
usage: halogen.py [-h] [-f FILE] [-d DIR] [-n NAME] [--png-idat] [--jpg-sos]

Halogen: Automatically create yara rules based on images embedded in office
documents.

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file  to parse
  -d DIR, --directory DIR
                        directory to scan for image files.
  -n NAME, --rule-name NAME
                        specify a custom name for the rule file
  --png-idat            For PNG matches, instead of starting with the PNG
file
                        header, start with the IDAT chunk.
  --jpg-sos             For JPG matches, skip over the header and look for
the
                        Start of Scan marker, and begin the match there.
```

Running the tool against the original files produced the following YARA rule.

```
$ python3 halogen.py -d ~/5901/halogen/
rule halo_generated: maldoc images
{
    meta:
        tlp = "amber"
        author = "Halogen Generated Rule"
        date = "2021-01-24"
        md5 = "['12453bda42e40501077b1d202a338648',
'138e3164dd69df539809a0bc5cd37d36', '678aff9ae66a0fb94043261eeeecaec5',
'5e870ca9e50adec114a98eab76f5b49c', 'b1d4a944e6a71d17037664135b3498eb']"
        family = "malware family"
        scope = "['detection', 'collection']"
        intel = "['']"
    strings:
        $jpg_img_value_0 =
{ffd8ffe000104a4649460001010100600060000ffe1004b4578696600004949 2a0008000000
0100988202002700000001a0000004100000047786e67643867613575333 95a746a73343462616
a346a743243376b7837}
```

```
        $png_img_value_1 =
{89504e470d0a1a0a0000000d4948445200000331000002300802000000039df72c60000000173
52474200aece1ce90000000467414d410000b18f0bfc6105000000097048597300000ec300000
ec301c76fa864000058}
        $png_img_value_2 =
{89504e470d0a1a0a0000000d49484452000003f900000444080600000002eb6d1290001000049
444154789cecfd698c26499adf89fdcddffb8afbc8c8fbacfbaeeaee6a9143ce0ae4f4ae280dc
9e10ca195b01c8a8016}
        $png_img_value_3 =
{89504e470d0a1a0a0000000d49484452000005bc0000069c08020000005800cfb20001000049
444154789cecfd69905de9991ff8bd67b9fbbe2fb9ef4820b1d48a228a2c924db2a9aa56abbb2
5b97ba615e376686cd2}
        $png_img_value_4 =
{89504e470d0a1a0a0000000d49484452000004ce000018608020000000cd85aa40000012669
43435041646f6265205247422028313939382900028cf6360603270747172651260 60c8cd2b2
90a72775288888c5260}

    condition:
        any of them
}
```

When using the Halogen generated rule, positive detection is confirmed against the initial file set. Additionally, running the derived YARA rule against a VirusTotal Retrohunt operation matched over 1300 samples with the same embedded graphical asset. A truncated list of 50 MD5 hashes is included in Appendix B. Using this approach for threat detection within an organization's defensive posture does require the initial acquisition and curation for threats to maintain the following rule.

```
$ yara ../halo_generated:maldoc_images .
halo_generated ./12453bda42e40501077b1d202a338648
halo_generated ./138e3164dd69df539809a0bc5cd37d36
halo_generated ./b1d4a944e6a71d17037664135b3498eb
halo_generated ./678aff9ae66a0fb94043261eeeecaec5
halo_generated ./5e870ca9e50adec114a98eab76f5b49c
```

7. Conclusion

The rise of document-based malware has driven the necessity of effective measures to prevent, detect, and respond to this attack vector. When considering the colloquial proverb

"Prevention is ideal, but detection is a must," innovative approaches are being developed to detect these threats. Regarding maldocs, they have often been accompanied by coercive lures to social engineer the victim into enabling the dynamic content.

The approach of using OCR-derived test in conjunction with a tailored YARA rule to alert on this type of activity has been explored, with some effort to evade the positives by manipulating the coercive lures. Through the research, various success was identified in achieving this objective, but some additional detection techniques were explored to supplement the OCR-based detection.

References

Amini, P., & Remen, M. (2019, September 30). Adobe XMP: Tales of an Overlooked Anchor. *https://inquest.net/blog/2019/09/30/Adobe-XMP-Tales-ofan-Overlooked-Anchor*

Eaton, K. & Roersma, W. (2020). Halogen (Version 1.2) [Software]. Available from *https://github.com/target/halogen.*

Krawetz, N. (2013). Kind of like that. *URL: http://www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html.*

InQuest.net. (2020). Malware Lures Gallery. Retrieved from *https://inquest.net/malware-lures-gallery*

Ogden, H., Sayre, K., & Roberts, C. (2018). MS Office File Formats – Advance Malicious Document (Maldoc) Techniques. Retrieved from *https://medium.com/walmartglobaltech/ms-office-file-formats-advanced-malicious-document-maldoc-techniques-b5f948950fdf*

Islam, N., Islam, Z., & Noor, N. (2017). A survey on optical character recognition system. *arXiv preprint arXiv:1710.05703*

Smith, J. (2020). Persian Kitties Hiding Benign Executables. Retrieved from *https://inquest.net/blog/2020/08/15/Persian-Kitties-Hiding-Benign-Executables*

Stevens, D. (2018). Didier Stevens - oledump.py. [online] Available: *https://blog.didierstevens.com/programs/oledump-py/.*

Stroshchein, J. (2020). Malware Samples: Maldoc templates. GitHub. Retrieved from *https://github.com/jstrosch/malware-samples/tree/master/maldoc_templates/2020/abuse_ch*

Tesseract. (2020). Tesseract OCR. GitHub. Retrieved from *https://github.com/tesseract-ocr/tesseract*

VirusTotal. (2021). b93c1b5898ee3d02d1f7996c90256099. Retrieved from *https://www.virustotal.com/gui/file/3c31bb37840e2413d56aec6497cf8f17a03dfd919713142a0937cbc2ce864fda/detection*

Appendix A

Link to analyzed images and maldoc samples for download

https://github.com/JosiahRaySmith/STI_OCR_Maldoc_Samples

Appendix B Halogen- detected samples (MD5)

```
0327a4558ea7b4401f772e2e5148c0ff
12453bda42e40501077b1d202a338648
138e3164dd69df539809a0bc5cd37d36
195b3a109bd3c9276026507138f6e954
1d7798702eb8689a080cb38a2c948ec1
1df0b5bc020b7debcd01a3634d2ece0f
20a938f804dd6ee995ee9cd923d75899
295a704c7ee659e745ddc526dbfc625b
2c15ede996f60f1f28b345ec923eca44
2f716680c59dde3a0343adf0667d1712
39b3250a21f54fdd8388f94e4a0d4932
3a9e194928c76e7052820563cd1a9f50
3ab2daf32a000abd8a5dc328235f51f9
3fe7bf47c10b57fd63d883f87b2e65ee
53dbb5386352cc5f3c370cd12c7d4cc3
57a2bfbcefd10610e9c0b2a5784fdf0b
5cad21a4eed485d9b56afa3857fe664f
5e870ca9e50adec114a98eab76f5b49c
5ed70d26731be613389933b056add082
632b845141ca60883026b2bf5ca0082e
678aff9ae66a0fb94043261eeeecaec5
6b5a3f7f2ddaca3b8905c0b9425fd4b4
726dbc2bfff09b63d781bb988a8f2e7c
748c844ad700769b4382496d3a3d2a5c
74f241d54d4213229e29a105bd3f5582
750c4a8653659ef775a4160cdfc2d0cf
77e0fb05daf8f607fb2e2b0d355ce5dd
79faaf40417d9217adf5509de9d2e9e9
842ea5f93523fab661d1aa55d49ff0d8
84ca0a804bb423e9fb0aece2592d63fb
8e15ea8a6bc8f38cc0f2643bcd84ae6c
9c3489eabf0e7723726ae3d0051717b9
ac5a832d975d248db09bcf61198a0e76
b1d4a944e6a71d17037664135b3498eb
bab8427df24379ef9696ffe14926cb9f
bc8ca1e5ddcf62b3f11a34278fe80d8a
be400144480ff3d883221711914fce1d
c6d2e97dff1b5d070d98e1b5249cd6ee
c86077cbded356552749da8097c4b9fb
d1e2fc34470e6c9519de9c608a9e13aa
d37d951eddb8b4d493814247ef3b05d6
d4c8c4161ba80890d32754290892504b
d5f2b5a1c76978cf58f6fa7287f503f6
d608c31c16d273efce98c2f99c45f858
e50c4fd673fc0890e212f3089dd9adbd
e510869a65f3654c8293ef2c10eeaad9
e5bd04c2680ed41ee962b85d56f11366
f3157b345a61c18ec4489005f70f44bc
feee9a27c8823963705749ec536cd5a4
```