Extracting Sections From PDF-Formatted CTI Reports

BEN DE KONING, University of Twente, The Netherlands

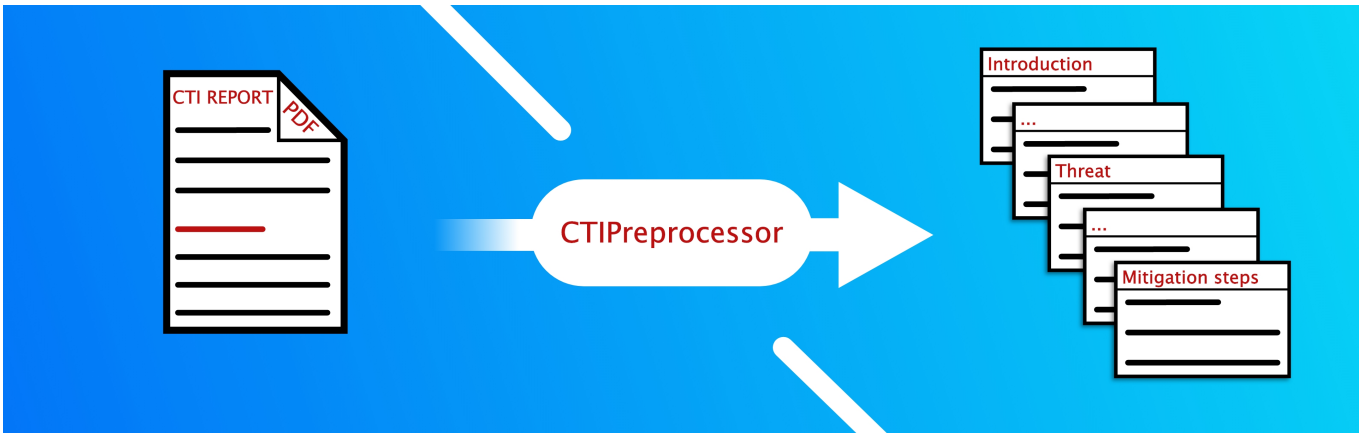

Fig. 1. A graphical display of how CTIPreprocessor takes a PDF-formatted CTI report and returns the different sections of the CTI report.

Extracting text from a PDF file is a task that sounds easier than its real-life execution. PDF files namely only know the position of the characters on the page, not knowing that the characters form words together. Another challenge is to separate it into different sections and paragraphs. The text and section extraction is important for pre-processing Computer Threat Intelligence (CTI) reports. Processing these reports is part of the task description of Security Operation Centers (SOCs). These reports contain valuable information on active cyber threats and are therefore important for cybersecurity. This research paper focuses on text extraction from a PDF-formatted CTI report, intending to extract the text separated into the sections present in the CTI report. This paper presents, after a thorough analysis of multiple candidate tools, which text extraction tool is preferred for text and section extraction from a PDF file using Python. This tool is then implemented to work on a real-world CTI report.

Additional Key Words and Phrases: PDF, CTI Report, SOC, Text extraction, Python, PDFPlumber, PyMuPDF, PyPDF2, Levenshtein

1 INTRODUCTION

Cyber Threat Intelligence (CTI) reports consist of incidents found by a Security Operation Centre (SOC). SOCs are responsible for the security of the IT infrastructures of businesses. CTI reports are important for SOCs since they provide a lot of information on a security alert, from background information to mitigation steps. This information can then be used by others to improve their cybersecurity systems and thus preventing a possible cyberattack or a virus from entering their systems.

Processing CTI reports manually is a time-consuming and inefficient task[11]. Moving from manual processing to automated processing would be an important improvement in the workflow of SOCs. By using less time, costs per CTI report could go down, more CTI reports could be processed per day, and processed CTI reports allow for improved working efficiency because of functionalities like searching and filtering. However, automated processing requires the reports to be pre-processed so that a computer will be able to understand the report. This pre-processing is necessary since the documents are written in natural language, while the computer would need raw text. CTIPreprocessor is a tool in development with the goal to automate the pre-processing of CTI reports from various sources on the internet. CTIPreprocessor will be able to process two different report formats: HTML and PDF. This project will be focused on PDF-formatted CTI reports.

Pre-processing CTI reports formatted as PDF is quite difficult due to the way PDF files are structured. PDF files do not work with words and paragraphs, but with the locations of the characters, making it a layout-based format[10]. This makes retrieving information from PDF files difficult, especially when structural details are to be retrieved as well, e.g., section and paragraph details. This project aims to solve the difficulty of pre-processing PDF-formatted CTI reports for CTIPreprocessor.

The goal of this research is to explore how to extract text from a PDF-formatted CTI report without losing document details (e.g., paragraphs, sections, etc.). To achieve the goal of this research, the following research question has been defined:

- **RQ** - To what extent is it possible to extract text from a PDF-formatted CTI report without losing document details, e.g., paragraphs and sections?

To help answer the research question, the following research sub-question has been defined:

- **RSQ** - What text extraction techniques can be used to extract text from a PDF document?

TScIT 37, July 8, 2022, Enschede, The Netherlands

This research is designed as a quantitative research. During the evaluation, a quality metric and a duration metric are used to determine the preferred text extractor from the list of candidate tools. This research started with determining the text extractors that are evaluated during this research (see Section 2). When the list of text extractors is complete, a theoretical analysis of the documentation of each text extractor is conducted (see Section 3). This analysis aimed to determine the theoretical functionalities of the text extractors. After the theoretical analysis, an evaluation of each tool is conducted within a controlled environment (see Section 4). This environment consists of 100 generated PDF files, each containing four predetermined sections with randomized but controlled text. See figure 3 for an overview of the evaluation process. Next, the preferred text extractor will be implemented on an actual CTI report, allowing an evaluation of its real-world capabilities (see Section 5). Finally, the discussion of the research results (see Section 6) and the conclusion (see Section 7) are presented. In the appendix (see Appendix A) are graphs belonging to the evaluation located.

2 CANDIDATE TOOLS

To determine when a text extractor is a usable and valid solution for CTIPreprocessor, it is good practice to create a list of requirements that a text extractor must fulfill to be a candidate for CTIPreprocessor. This list contains two distinct categories of requirements: mandatory requirements and preferred requirements. The mandatory requirements are:

- Able to extract text from a PDF-formatted file with the text not displayed within an image
- Able to be used with Python3
- Using this tool, being able to identify sections
- Using this tool, able to identify paragraphs

The preferred requirements are:

- The tool is open source
- Able to select which parts of the document you want to extract

Next, a search for candidate text extractors was started. This search took place on the internet, using Google Search and forums like StackOverflow[2] and Codegrepper[8][9]. From this search, the following candidate text extractors have been found:

- **PDFPlumber** - After finding the GitHub repository of this tool[12], a quick read of the description showed that this tool is capable of text extraction from a PDF file, consists of 99.5% of Python code, and can extract detailed information about the characters, including the font and font size. The font and font size can be used to determine different sections within a PDF file. Also, PDFPlumber is open-source, and when using the location of the characters, the ability to select which parts of the document you want to extract could be realized as well.
- **PymuPDF** - After finding the PyMuPDF GitHub directory[13], it describes the tool as a connection of the original tool MuPDF[5] to Python. A quick read of the description reveals that PyMuPDF allows for the extraction of text from a PDF file preserving the layout of the document. This feature description could allow for the extraction of sections and paragraphs. The tool can also be used in Python, although it is mostly written in SWIG.
- **PyPDF2** - In the documentation of PyPDF2[1] is mentioned that the library can extract text from a PDF file, although this is a difficult thing to achieve. There is no other information about its capabilities, only that it is pure-python and capable of extracting metadata as well. Altogether, it made it worth it to have PyPDF2 in the list of candidate text extractors for CTIPreprocessor.

3 THEORETICAL ANALYSIS

3.1 PDFPlumber

PDFPlumber's documentation[12] analysis reveals that PDFPlumber is capable of two different text extraction versions: with layout and without layout. The layout refers to the layout of the words on the page of the PDF file, allowing PDFPlumber to copy this layout and space the words and sentences so the layout of the text extraction is similar to the layout of the PDF file. For example, when the PDF file has the date written in the top right corner of the page, the layout functionality of PDFPlumber would space the words in the text extraction in such a way that the date is also in the top right corner of the result. When the layout functionality is disabled, the text extraction simply extracts the text, without focusing on the layout.

For text extraction, the following four parameters can be altered to influence the text extraction:

- **X_tolerance** - Altering this parameter can result in more or fewer spaces between characters
- **Y_tolerance** - Altering this parameter can result in more or fewer newline characters between characters.
- **X_density** and **Y_density** – Altering these parameters can result in more or fewer characters or newlines per point, with point referring to the PDF unit of measurement. This is used when the layout functionality is active.

It is also possible to extract the words with their details, like location and font size. This could be an important functionality for the extraction of the different sections since the font size can be used to determine the headers.

Other functionalities include the extraction of tables and the creation of an image of the PDF file's pages that have rectangles surrounding each detected word.

3.2 PyMuPDF

PyMuPDF has multiple different text extraction options for extracting text from a PDF[6]:

- **text** – This option extracts plain text without any formatting. This is the default option.
- **blocks** – This option extracts the text in blocks. If text is connected (e.g., a paragraph or a sentence), then they will also be combined into one block. This option returns the blocks as a list.
- **words** – This option will generate a list containing the words found in the PDF file. Spaces will be left out.
- **html** – This option transforms the PDF file into a fully working HTML file.

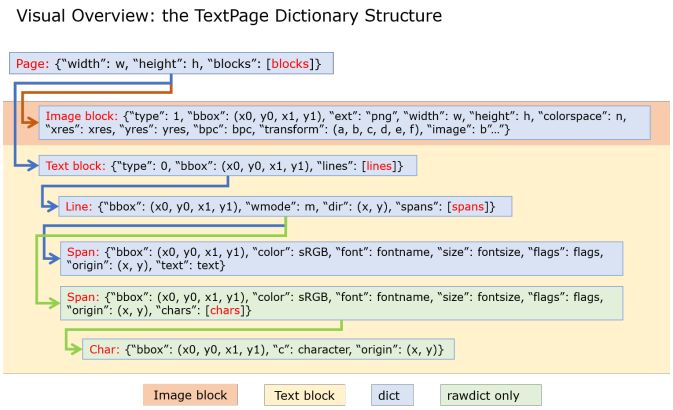

Fig. 2. Structure of TextPage object in PyMuPDF[7]

- **dict** or **json** – This option extracts the text as a dictionary or a JSON string. It is said to have the same information level as *html*.
- **rawdict** or **rawjson** – This option is a more detailed version of *dict* or *json*. It namely also includes character details.
- **xhtml** – This option extracts the text with a similar quality as *text* but also includes images and can be opened in a browser.
- **xml** – This option returns an XML object that contains positional and font information of each character.

From this list of text extraction options, *text*, *blocks*, and *html* are the most interesting. *text* can be used if only plain text is needed, *blocks* is mentioned to have the ability to be divided into paragraphs as shown in the PDF file, and *html* could be interesting since HTML code could be easier to extract sections from, depending on how the HTML is generated.

Next to the text extraction option, there is also the functionality of creating a so-called TextPage[7]. This object contains python dictionaries of a page of a PDF file. Each dictionary has another dictionary or list embedded, containing more detailed information. The structure of a TextPage can be seen in Figure 2. This functionality looks to be a powerful asset in determining the sections of a PDF file.

3.3 PyPDF2

PyPDF2's documentation[1] does not mention any extra features next to the plain text extraction. The PdfReader Class documentation does not mention any interesting features regarding text extraction. Also, the PageObject class only mentions the standard text extraction functionality, which does have some spacing parameters. PyPDF2 does have a vast list of other functionalities it can do with a PDF file, making it an interesting tool for other tasks than text extraction. Because of the limited time this project has and the documentation showing no sign of text extraction functionalities with unique features, PyPDF2 will not be included in the evaluation.

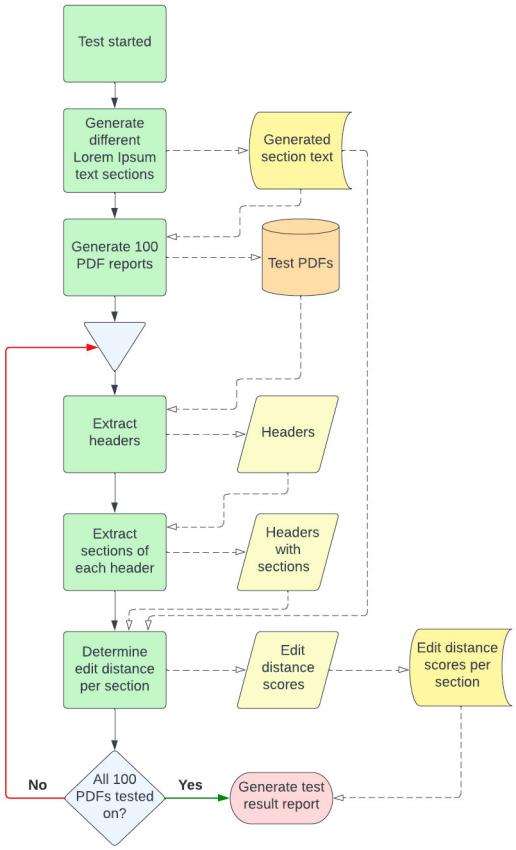

Fig. 3. The evaluation process.

4 EVALUATION

4.1 Methodology

For the evaluation, a controlled evaluation process was designed in Python. This process is visualized in Figure 3. During this evaluation, a test is defined as extracting the headers and sections, and determining the metrics for one PDF file. Thus, each evaluation has an equal number of PDF files as tests. The evaluation begins with preparation for the tests by first generating four different Lorem Ipsum texts, followed by generating 100 PDF files. each PDF file contains four different sections: *Introduction*, *Background Information*, *Threat Information*, and *Mitigation Steps*. The text of each section is a randomized choice between the four Lorem Ipsum texts. Also, the section *Threat Information* contains a lorem ipsum list of bullet points. This is added to analyze how such a list influences the results. With the PDF files ready, the text extractor is used on each PDF file. this subprocess starts with the extraction of the headers. This is done by filtering the font size. Next, the sections are extracted using the earlier extracted headers as dividers of the sections. Finally, the extracted sections are compared with the Lorem Ipsum text it originated from. This comparison is scored using the Levenshtein edit distance. When all one hundred scores have been collected,

the evaluation result report is generated. This report contains the following information about the evaluation:

- the full extraction result of the last PDF file. It is displayed per section, with the header above the text of the section, -
- the duration of the evaluation, the average edit distances per section, and the average edit distance in total, -
- and the individual edit distances collected during the evaluation.

By exporting the individual edit distances to Excel, the data is visualized in graphs with each graph showing the edit distances of a different section throughout the evaluation. These graphs can be found in the appendix (see Appendix A.1 and A.2).

Important to note is that the edit distance scores are only saved when the header of the section is correctly extracted. This means that having one hundred edit distances saved for a certain section, the header extraction works flawlessly.

4.1.1 Measurement tools. The accuracy scoring system used during this research is the Levenshtein edit distance between two given strings. This metric determines the number of steps it takes to convert the first string of text into the second one. Conversion is done with only three different tasks: deletion, insertion, and substitution[4]. The lower the edit distance is, the more similar the two strings are. When the edit distance equals zero, the two strings are considered identical. The library used is called the *Levenshtein* library.

The duration of the evaluation is determined using the *time* library. This library can determine the current time in high detail and allows calculations with it. By subtracting the starting time of the evaluation from its ending time, the duration is determined.

Other libraries used worth mentioning are the following:

- **from fpdf: FPDF, HTMLMixin** - This library is responsible for the creation of PDF files in Python. Using this library, one can create a PDF file with images, text, and cosmetic details. Locations and details of the various parts of the PDF file can all be customized, making it a powerful tool for automated PDF file creation.
- **pdfplumber** - This library is responsible for running the tool PDFPlumber.
- **fitz** - This library is responsible for running the tool PymuPDF.
- **from PyPDF2: PdfReader** - This library is responsible for running the tool PyPDF2.

4.1.2 Measurement environment. The evaluations are run on an HP ZBook Studio G4. This laptop is equipped with 32GB of memory, a 1TB SSD for storage, an Intel Core I7-7700HQ processor, and an Nvidia Quadro M1200M graphics card. The evaluations are not connected to or influenced by the internet connection, making the network quality unimportant. The laptop is running Windows 10 Home version 21H2. The evaluation is run in Python, with the version used during this evaluation being version 3.10.4:9d38120.

The evaluation environment is created in the programming tool called Microsoft Visual Studio Code (VSC). This tool allows for the creation of programming projects and the management of files within them. It also contains a market for extensions, allowing a programmer to use it for most programming projects.

```
RESULTS (repeats = 100)
=======
Section 'Introduction' scored 34.5
Section 'Background Information' scored 35.26
Section 'Threat Information' scored 108.28
Section 'Mitigation Steps' scored 39.14
Average score in total: 54.295

Duration of test: --- 47.8108491897583 seconds ---
```

Fig. 4. Results of the PDFPlumber test

The evaluation is conducted on 100 separately generated PDF files. The 100 PDF files have a combined size of 398 kB, which results in an average file size of 3.98 kB. Next to the PDF files, there is also one metadata.txt file, which size equals 9.57 kB for 100 PDF files. The result of the evaluation will be saved in a .txt file. The size of this document varies, depending on the PDF file size. The size of the evaluation result files can be expected to be around 11.5 kB.

4.2 Results

4.2.1 PDFPlumber. The PDFPlumber evaluation was completed in 47.8 seconds. the text extractor scored an average edit distance of 54.295, meaning that on average there are 54.295 steps to be taken for the text extraction of PDFPlumber to be equal to the original text. The average edit distance per section can be found in Figure 4. The lowest average edit distance equals 34.5 and is given to the section *Introduction*, while the highest average edit distance equals 108.28 and is given to the section *Threat Information* (note that a lower score is considered better). The graph containing the edit distances collected per section can be found in Appendix A.1. Each graph contains one hundred data points, which shows that the extraction of the headers has not failed once during this evaluation. Looking at the extracted text itself, it can be noted that PDFPlumber keeps the same sentence structure as the PDF. This means that when a row in the PDF ends, the text extractor adds a line break to the extracted text. The result does not show any signs of paragraph separation.

4.2.2 PyMuPDF. The PyMuPDF evaluation was completed in 2.90 seconds. the text extractor scored an average edit distance of 24.588, meaning that on average there are 24.588 steps to be taken for the text extraction of PDFPlumber to be equal to the original text. The average edit distance per section can be found in Figure 5. The lowest average edit distance equals 17.46 and is given to the section *Background Information*, while the highest average edit distance equals 42.76 and is given to the section *Threat Information* (note that a lower score is considered better). The graph containing the edit distances collected per section can be found in Appendix A.2. Each graph contains one hundred data points, which shows that the extraction of the headers has not failed once during this evaluation. Looking at the extracted text itself, PyMuPDF allows for the text to be extracted in paragraphs. Each paragraph is in one line, making it easy and convenient to extract the sections in paragraphs.

4.3 Comparison

When comparing the scores of both tools, it becomes clear that PyMuPDF has significantly better scores than PDFPlumber. The

```
RESULTS (repeats = 100)
=======
Section 'Introduction' scored 18.62
Section 'Background Information' scored 17.46
Section 'Threat Information' scored 42.76
Section 'Mitigation Steps' scored 19.51
Average score in total: 24.588

Duration of test: --- 2.9022371768951416 seconds ---
```

Fig. 5. Results of the PyMuPDF test

average edit distance scored during the evaluation of PyMuPDF is about three times lower than PDFPlumber. A similar trend is visible when comparing the average edit distances per section. When comparing the text extraction itself, it becomes clear that PDFPlumber does not show any functionality in separating paragraphs, while PyMuPDF separated each paragraph it detected and allows for easy paragraph extraction.

When looking at the graphs, it becomes clear that every section has one hundred data points, which shows that the extraction of the headers from the PDF files works flawlessly. Comparing the graphs show that the evaluation of PDFPlumber shows significantly larger fluctuations than the evaluation of PyMuPDF. This trend is expected to reflect the consistency of text extraction quality, showing that PDFPlumber has difficulty being consistent with the extraction quality.

4.4 Preferred Text Extractor

Based on the comparison of the evaluations of PDFPlumber and PyMuPDF, PyMuPDF is the preferred tool for usage in CTIPreprocessor. The main argument for this is that PyMuPDF has a significantly better score and time, next to the more convenient extraction of paragraphs. Also, the edit distances of PyMuPDF are more consistent over 100 generated PDF files than the edit distances of PDFPlumber.

5 IMPLEMENTATION

Now that the preferred text extractor for CTIPreprocessor is determined, it can be further implemented for usage on an actual, real-world CTI report. The report is collected from the website of the Cybersecurity & Infrastructure Security Agency (CISA) [3]. The following improvements were made:

- The text extractor will try to limit the text extraction to the middle area of the page, thus excluding the header and footer of the page. This prevents text equally sized as the header or paragraphs from extracting as well. This is still not fully working due to the difficulty of determining the borders between the header, footer, and middle parts. The measuring unit is not specified in the documentation.
- The extracted text is now actively filtered on excessive spaces and line breaks.
- The header extraction is now capable of extracting headers that are divided over two or more rows.
- Other small improvements of stability of the code.

After updating and improving the code, it was executed on the real-world CTI report, with the result exported in a .txt file. This resulted in the following noticeable differences from what is expected from the text extraction:

- Although the code tries to ignore the header, a sentence with the same font size as the headers was extracted as a header as well. This header does not have any text connected to it, however, making it easy to remove after extraction.
- On the first page, a text block called *Best practices* is placed on the right of the text. This resulted in the text extractor failing to successfully determine the paragraphs of the text. Luckily, the text block is not spread throughout the text but is added at the bottom of the text extraction of that section. This shows that the text extractor is capable of dividing multiple columns of text.

5.1 Future Work

For further versions of this text extractor of CTI reports, it is suggested to focus on the system for determining what is a header and what is text. Currently, the font sizes of the headers and the text are expected to be known before text extraction. Also, the system for ignoring the header and footer can use some improvements. Currently, the limits are hard-coded, while it would be better to have this automatically determined or at least able to be provided. Finally, the text extraction can be worked on for even better detection of paragraphs, although it currently works well as well.

6 DISCUSSION

Looking at the evaluation results of PDFPlumber and PyMuPDF, it becomes clear that the section *Threat Information* has significantly higher scores than the other sections. This trend is visible for both tools. A possible explanation for this is the addition of the bullet points to the section *Threat Information*, which is the only difference between this and the other sections.

The graphs also show another interesting trend, namely that the edit distance values per section are only four different values. This is best noticeable in the graphs of the PDFPlumber evaluation since these data points show a larger fluctuation in values. The reason for this trend is that there are four different section texts from which the PDF generator can choose.

The edit distance fluctuation of the PDFPlumber evaluation is also significantly larger than the edit distance fluctuation of PyMuPDF. This, however, can also be influenced by the difference in implementation of both tools. Although the implementation of both tools was aimed to be as similar as possible, the tools do not have equal capabilities. Therefore, the fluctuation is still a good representative of the consistency of text extraction quality.

7 CONCLUSION

Based on the evaluation results of this research, it can be said that it is possible to extract text from a PDF-formatted CTI report without losing document details. However, it must be noted that it would require noticeably more time and effort that this research project allowed. Also, perfect extraction of document details will be hard to accomplish, although the results can be close to perfect.

There are numerous tools available that all use a slightly different technique for extracting text from a PDF file. Based on the results of this research paper, PyMuPDF is the preferred tool for text extraction using Python, followed by PDFPlumber.

REFERENCES

[1] 2005. Welcome to pypdf2. https://pypdf2.readthedocs.io/en/latest/index.html
[2] 2016. How to extract text from a PDF file? https://stackoverflow.com/questions/34837707/how-to-extract-text-from-a-pdf-file
[3] 2022. Alert (AA22-158A). https://www.cisa.gov/uscert/ncas/alerts/aa22-158a
[4] 2022. Levenshtein distance. https://en.wikipedia.org/wiki/Levenshtein_distance
[5] 2022. MuPDF Overview. https://mupdf.com/
[6] 2022. PyMuPDF documentation. https://pymupdf.readthedocs.io/en/latest/
[7] 2022. TextPage. https://pymupdf.readthedocs.io/en/latest/textpage.html#textpage
[8] 2022. the hacker man's Profile. https://www.codegrepper.com/profile/zeke-john
[9] 2022. XeNN0N's Profile. https://www.codegrepper.com/profile/priyam-harsh
[10] Hannah Bast and Claudius Korzen. 2017. A Benchmark and Evaluation for Text Extraction from PDF. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 1–10. https://doi.org/10.1109/JCDL.2017.7991564
[11] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. 2017. TTPDrill: Automatic and Accurate Extraction of Threat Actions from Unstructured Text of CTI Sources. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (Orlando, FL, USA) *(ACSAC 2017)*. Association for Computing Machinery, New York, NY, USA, 103–115. https://doi.org/10.1145/3134600.3134646
[12] Jsvine. 2016. Jsvine/pdfplumber: Plumb a PDF for detailed information about each char, rectangle, line, et cetera and easily extract text and tables. https://github.com/jsvine/pdfplumber
[13] Pymupdf. 2016. Pymupdf/pymupdf: Python bindings for mupdf's rendering library. https://github.com/pymupdf/PyMuPDF

A GRAPHS OF TEST RESULTS

A.1 PDFPlumber

Figure 6, Figure 7, Figure 8, and Figure 9 show the edit distance scores collected during the PDFPlumber evaluation, with every figure showing a different section.

A.2 PyMuPDF

Figure 10, Figure 11, Figure 12, and Figure 13 show the edit distance scores collected during the PyMuPDF evaluation, with every figure showing a different section.

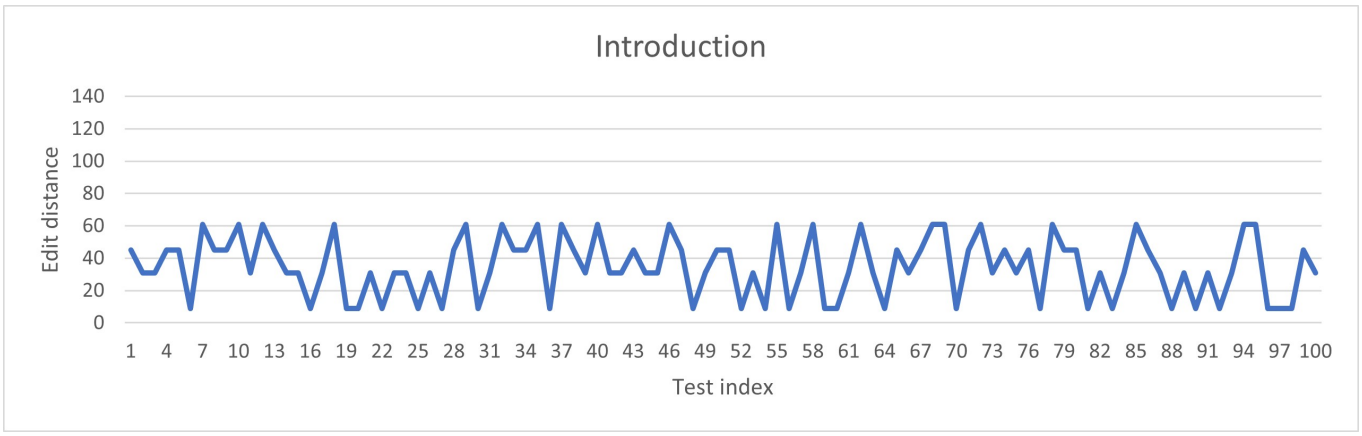

Fig. 6. Scores of PDFPlumber test for the section 'Introduction'

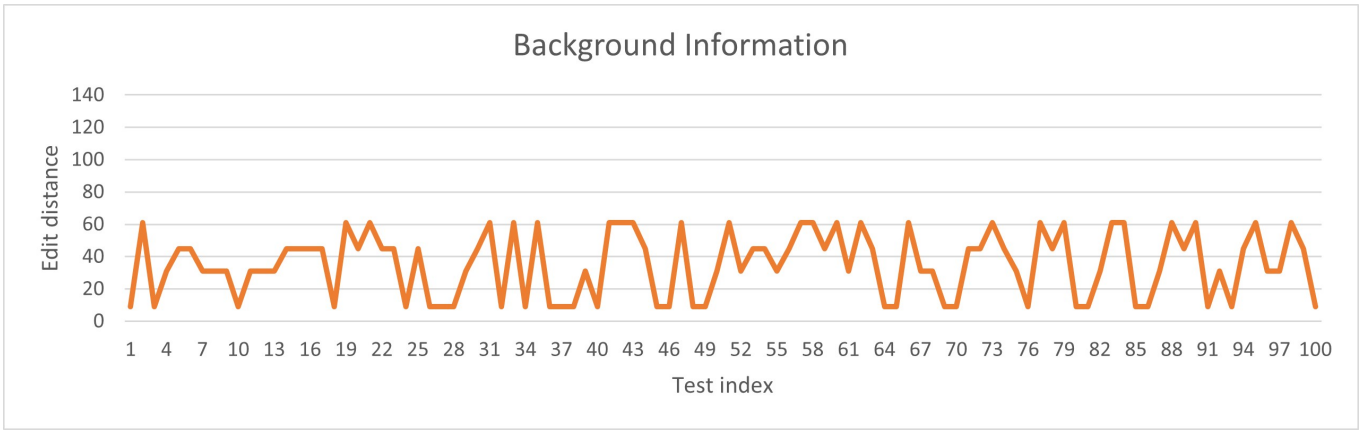

Fig. 7. Scores of PDFPlumber test for the section 'Background Information'

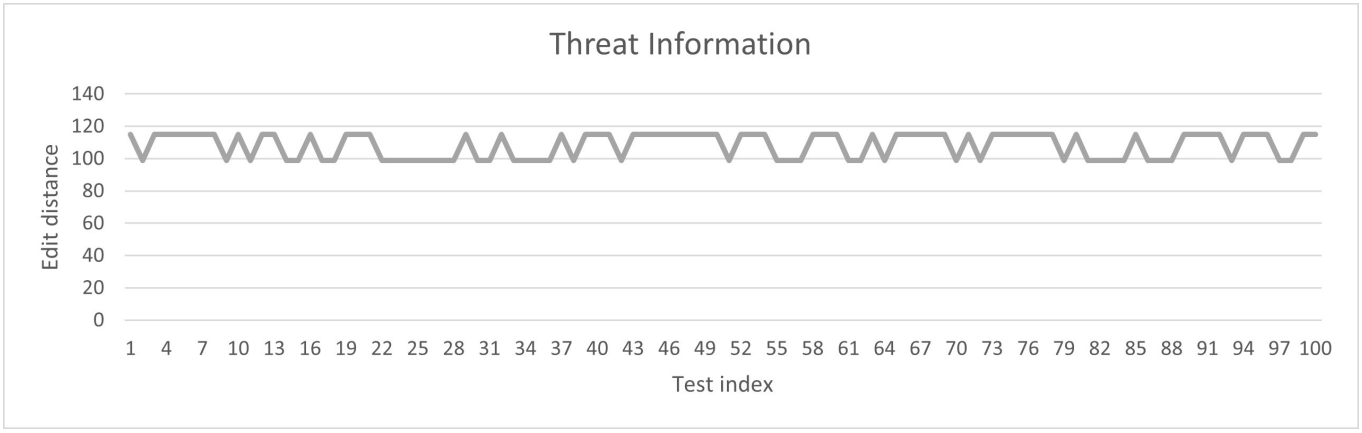

Fig. 8. Scores of PDFPlumber test for the section 'Threat Information'

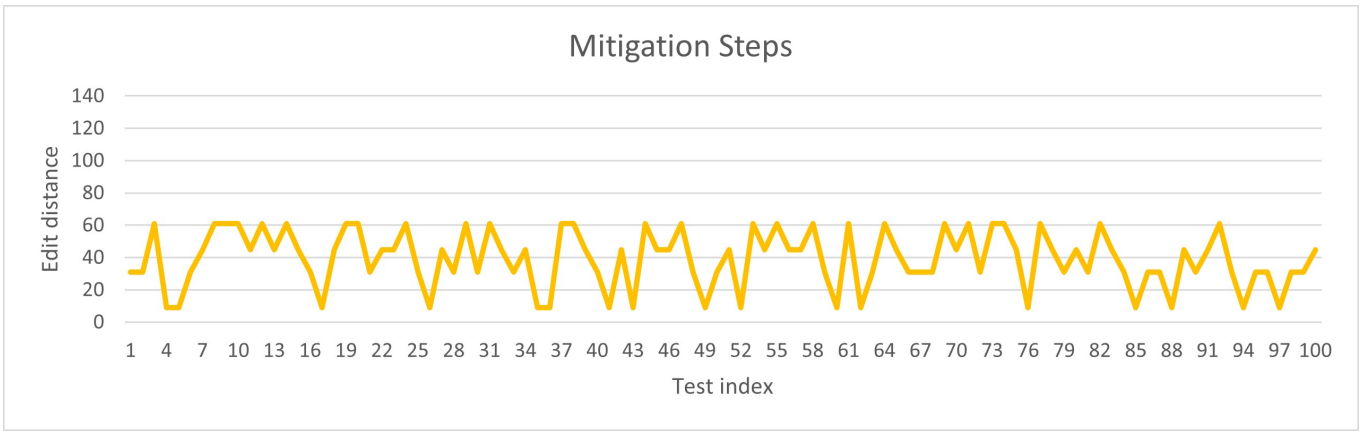

Fig. 9. Scores of PDFPlumber test for the section 'Mitigation Steps'

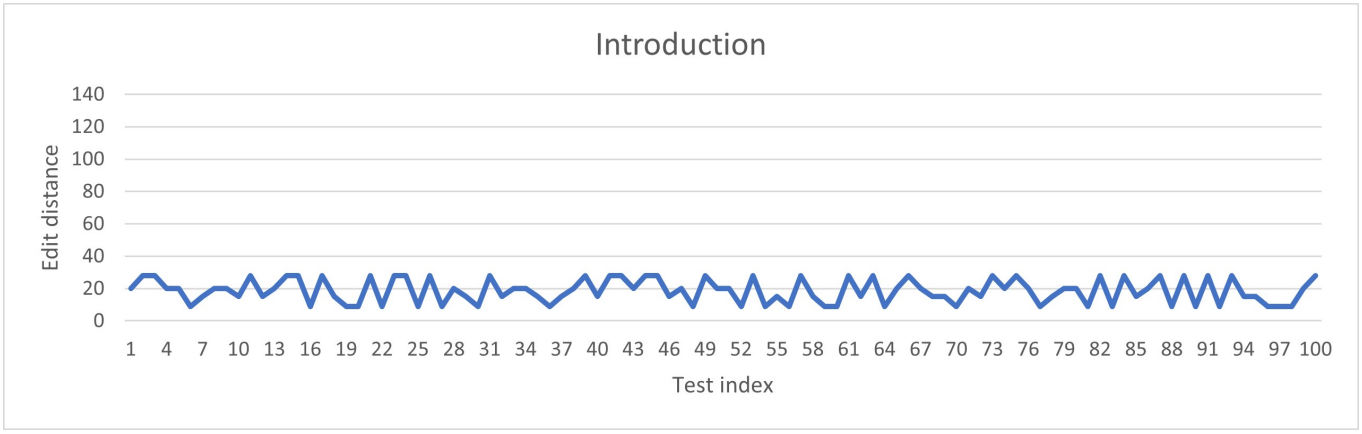

Fig. 10. Scores of PyMuPDF test for the section 'Introduction'

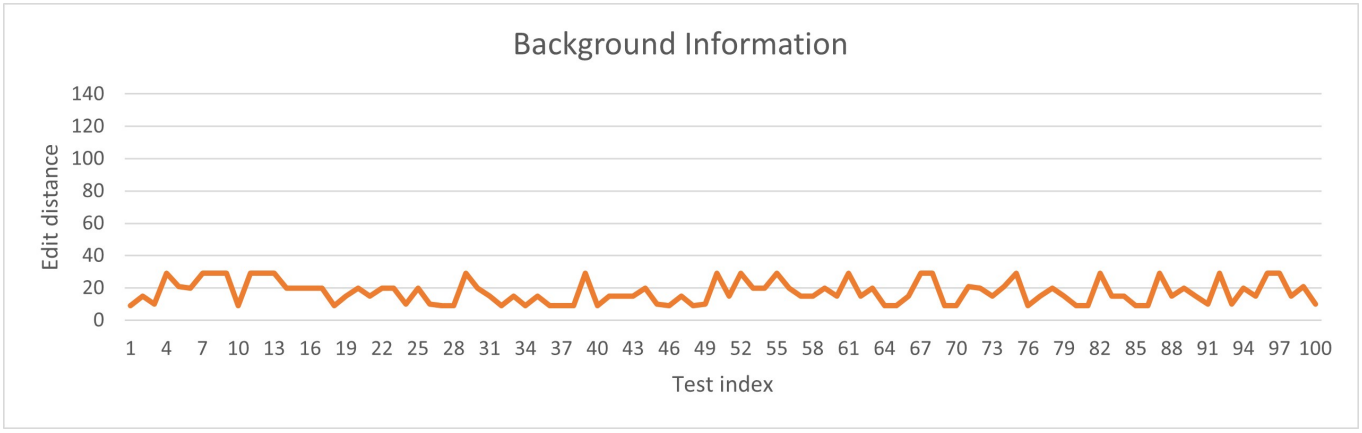

Fig. 11. Scores of PyMuPDF test for the section 'Background Information'

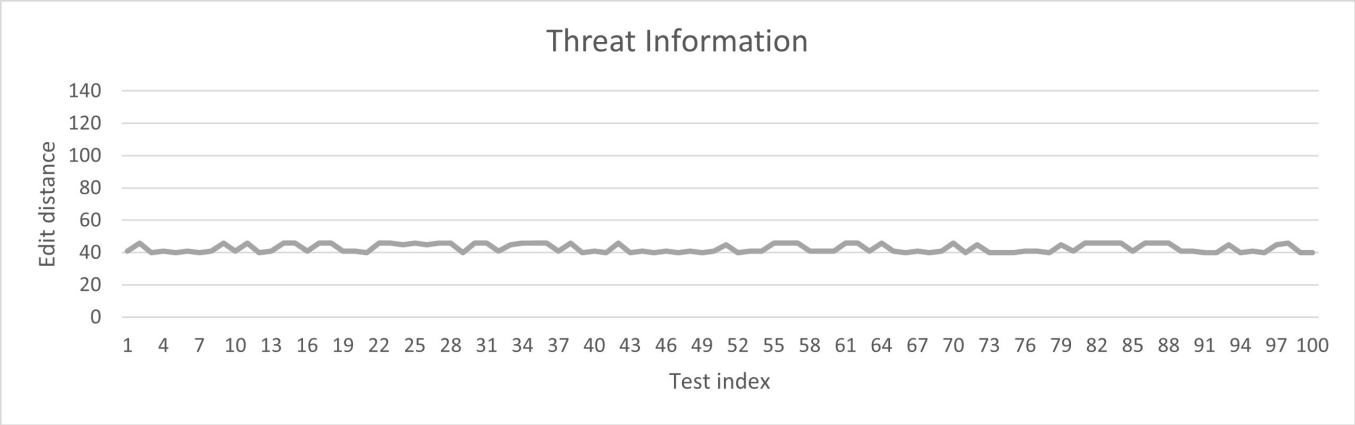

Fig. 12. Scores of PyMuPDF test for the section 'Threat Information'

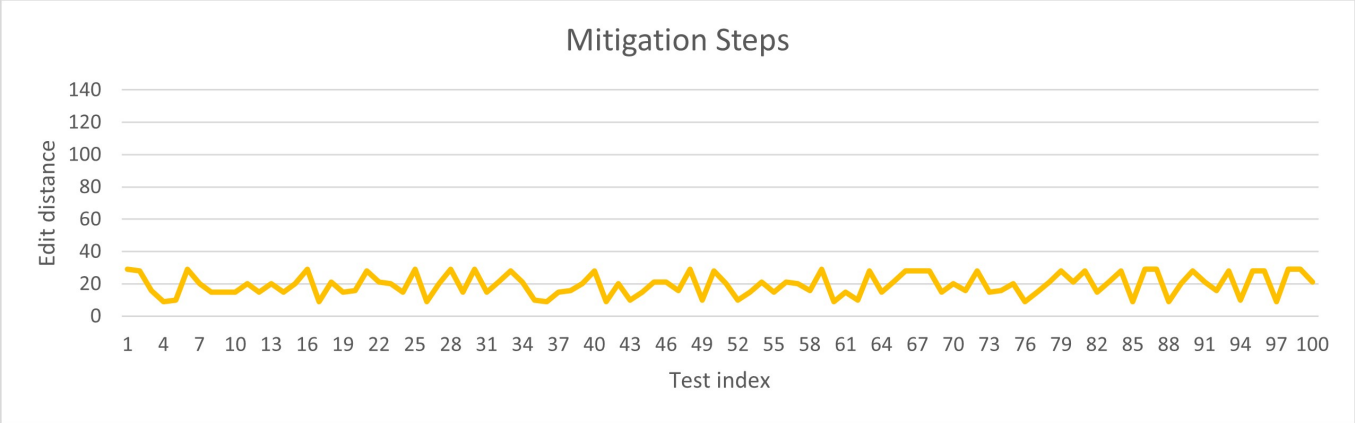

Fig. 13. Scores of PyMuPDF test for the section 'Mitigation Steps'