Research Report: Building a File Observatory for Secure Parser Development

Tim Allison*, Wayne Burke†, Chris Mattmann‡, Anastasija Mensikova§, Philip Southam¶,
Ryan Stonebraker‖,
Jet Propulsion Laboratory, California Institute of Technology
Pasadena, California
*timothy.b.allison@jpl.nasa.gov, †wayne.m.burke@jpl.nasa.gov, ‡chris.a.mattmann@jpl.nasa.gov, §anastasia.menshikova@jpl.nasa.gov,
¶philip.southam@jpl.nasa.gov, ‖ryan.a.stonebraker@jpl.nasa.gov,

***Abstract*—Parsing untrusted data is notoriously challenging. Failure to handle maliciously crafted data correctly can (and does) lead to a wide range of vulnerabilities. The Language-theoretic security (LangSec) philosophy seeks to obviate the need for developers to apply *ad hoc* solutions by, instead, offering formally correct and verifiable input handling throughout the software development lifecycle. One of the key components in developing secure parsers is a broad coverage corpus that enables developers to understand the problem space for a given format and to use, potentially, as seeds for fuzzing and other automated testing. In this paper, we offer an update on the development of a file observatory to gather and enable analysis on a diverse collection of files at scale. Specifically, we report on the addition of a bug tracker corpus and new analytic methods on our existing corpus.**

***Index Terms*—LangSec, language-theoretic security, file corpus creation, file forensics, text extraction, parser resources**

I. Introduction

Software that processes electronic files is vulnerable to maliciously crafted input data. Language-theoretic security (LangSec) is one software development method that offers assurance of software free from common classes of vulnerabilities. Whether LangSec parsers are built from formal specifications or are derived from samples, these parsers require wide-reach corpora for inference and/or integration testing throughout the development cycle.

In our LangSec 2020 paper [1], we reported our findings in setting up an initial file observatory to support file research for parser developers. In this paper, we report on our new directions in corpus gathering and progress in feature extraction and analytics. As with our earlier paper, our focus is still on the Portable Document Format (PDF), but we have expanded our efforts to gather a greater variety of formats.

As before, we believe that our lessons learned and work to-date will help address some of the challenges faced by researchers and parser developers who need to generate and analyze their own corpora.

The research was carried out at the NASA (National Aeronautics and Space Administration) Jet Propulsion Laboratory, California Institute of Technology under a contract with the Defense Advanced Research Projects Agency (DARPA) SafeDocs program.

II. Background and Related Work

Garfinkel *et al.*'s led the way with GovDocs1 [2] as a critical, large scale collection of documents for forensics researchers to use to develop and test their tools. Since then, in the open source software world, at least three Apache Software Foundation projects (Apache Tika [3], Apache PDFBox [4] and Apache POI [5]) rely on Garfinkel *et al.*'s corpus for large scale regression testing and have extended this corpus to include a richer set of more diverse and more recent file types [6] [7]. These corpora enable direct comparison of different approaches and tools under development, in addition to providing the means for reproducibility [2] [8] [9].

III. Data Sources

As we reported last year, we started by gathering 20 million unique PDF files from Common Crawl [10]. Common Crawl crawls hundreds of terabytes of data from the web each month and makes the data readily available on Amazon Web Services' public datasets. Common Crawl greatly simplifies the task of crawling the web. We made around four million of these files available to performers and the evaluation team. We found that processing this scale of files was prohibitive for some team members, and even for those with cloud resources, it was extremely inefficient to process that many files. We sought a way to distill these files to those that were problematic with fewer resources.

One of our colleagues, Peter Wyatt, a board member of the PDF Association, recommended that we run a targeted crawl of open source bugtracker sites on the hypothesis that these sites will have a much higher proportion of "stressful" PDFs – files that caused crashes or other denial of service vulnerabilities – than the general internet.

In the following sections, we summarize our findings with Common Crawl and with our new bug tracker corpus.

A. Common Crawl

Common Crawl remains the the primary source for much of our work. In [1], we noted that Common Crawl offers an efficient way to obtain and process a large amount of documents. The two main caveats we identified remain true:

1) Common Crawl truncates files at 1 megabyte (MB) meaning that users may have to refetch files from the original URL to obtain the full file
2) Common Crawl represents effectively a convenience sample of the web:
    a) those sites that are easily reachable and crawlable are well represented; those sites that are remote, make heavy use of javascript or do not contain a rich internal link structure may not be as well represented
    b) file formats and subtypes of formats that do not tend to appear on web pages will be rare or entirely absent

Nevertheless, the utility of Common Crawl data for file research is unparalleled.

While documentation and proper packaging remains to be added, we have made our Common Crawl code available on github: https://github.com/tballison/file-observatory/tree/main/commoncrawl-fetcher.

B. Bug Tracker Corpus

As mentioned in the introduction, this year we sought a way to target an internet crawl to gather a condensed set of stressful PDFs. Based on Peter Wyatt's suggestion to focus on open source bug tracker sites, we wrote custom crawlers for common bug tracker application programming interfaces (APIs), including github, JIRA, bugzilla and gitlab. Where possible, these crawlers used the existing APIs available for these different bug tracking packages. In a few cases, however, where APIs were not available, we wrote custom HTML scrapers to iterate through issues and download attachments. We have made this code available: https://github.com/tballison/tika-addons/tree/main/bugtracker-crawlerhttps://github.com/tballison/tika-addons/tree/main/bugtracker-crawler.

In addition to crawling the sites to gather attachments, we identified two other critical components:

1) many users package and/or compress their example files in tar+gzip files or zip files (for example). We wrote code to unpackage and decompress attachments.
2) on some bugtracking sites, it was customary for users to post external hyperlinks instead of attaching files. We added processing to download files from external links.

Given our initial focus on PDFs, we identified 35 issue trackers that were likely to contain stressful PDFs. These represented the issue trackers for 32 different tools – three of the tools had migrated from Bugzilla to gitlab over time, yielding two bug repositories for each of those tools. As Peter Wyatt documents [11], these tools were based on a wide variety of languages, including java, c/c++, go, javascript and python.

In our November 2020 crawl of bugtracker sites, we gathered roughly 1.2 million files, comprising 311 gigabytes (GB), from the 35 bug trackers. We collaborated with the Apache Tika [3] to make these files available: https://corpora.tika.apache.org/base/docs/bug_trackers/.

From this larger corpus we created a PDF-centric subset of roughly 33,000 PDF files comprising 31GB. We packaged these files for PDF developers and made them available: https://corpora.tika.apache.org/base/packaged/pdfs/pdfs_202011/.

We ran numerous open source and a few proprietary tools against this PDF-centric subset and notified developers of potential problems, for example: https://github.com/pdfcpu/pdfcpu/issues/246. Commercial parser developers have privately expressed gratitude for this corpus because it uncovers numerous denial of service vulnerabilities.

In addition to outreach, we wanted to quantify how "stressful" this corpus was compared to a control corpus. We gathered a "control corpus" of 30,000 PDFs randomly selected from Apache Tika's regression corpus, a combination of govdocs1 [2] and Common Crawl. We selected Apache Tika as one tool to run against both corpora. Given that Tika and its dependencies have started using this corpus as part of its larger regression testing process, we expected that we might not find significant differences between the bugtracker corpus and the control corpus. We were wrong.

As one metric, we counted the number of times that Tika had to restart. Tika restarts on catastrophic failures, including infinite loops, stack overflows, and out of memory errors – these are serious denial of service issues. On the bugtracker corpus, Tika had to restart 13 times; on the control corpus, Tika did not have to restart at all.

As another metric of "stressful"-ness, we counted more common parse exceptions. There were 519 parse exceptions in the bugtracker corpus (1.6%) compared with 16 in the control corpus (0.05%).

In short, this methodology and this corpus, in particular, offers a highly condensed set of stressful files.

IV. Data Analysis

In the following three sections, we outline three different thrusts to support analytics on this large corpus: a) identifying variants and hierarchical anomalies in the data, b) a user interface to enable simpler interaction with the data, c) preliminary investigations into machine learning to automatically classify the creator tool for PDFs based on features within the files.

A. (Mis?)Using Search Components in Elasticsearch

As we worked with PDF experts, we identified two critical needs for analytics. First, they wanted a way to quickly identify variants or common misspellings. Second, they wanted to be able to quickly quantify parent-child relationships in the PDF Document Object Model (DOM). For example, if the parent object is a '/Resources' object, what are the keys in its child object and how often do they occur.

For the first challenge, we found that we could use ElasticSearch's Term Suggester feature with minimal amounts of post processing. This is typically used in search engines to recommend "correct" spellings. We found that it uncovered some interesting variants, as we show in 1. One PDF expert was already aware of '/SubType' as a common misspelling for

/Subtype 8798	
/Subtype	8798
/SubType	41
/Subtype2	4
/subtype	1
/CapHeight 6489	
/CapHeight	6489
/CapHieght	8
/CVHeight	2
/ColorSpace 5748	
/ColorSpace	5748
/Colorspace	1
/FirstChar 5650	
/FirstChar	5650
/FirtsChar	1
/Widths 5646	
/Widths	5646
/Width 4625	
/WXdths	1

Fig. 1: Interesting spelling variants for common keys in PDFs.

'/Subtype' (keys should be case sensitive in PDFs), but we now offered a tool to discover other misspellings automatically.

For the second challenge, we used Elasticsearch's 'auto-complete' feature to enumerate the terms that included, e.g. '/Resources-¿', such as '/Resources-¿Colorspace', as in 4b.

B. File Observatory Interface

In order to assist in understanding the depth of the file observatory corpus, a tool was needed to properly explore it. Kibana exists for this purpose as a generic tool for exploring Elasticsearch indices, however, it proved to be lacking in several critical areas that were specific for this application such as the ability to download specific PDFs that are stored separately or to initiate chained Elasticsearch queries to perform more complex aggregations. To address these issues, a custom file observatory web application was created.

The goal of the file observatory web application was threefold; to allow generic searching and filtering of the file observatory in a similar manner as kibana, to allow PDFs to be downloaded either individually or en masse, and to allow for custom visualizations and metrics that would otherwise not have been possible.

1) Searching and Filtering: The file observatory index exists to provide metadata on the stored PDFs in an as-needed capacity. As such, it exists in a partial state of flux as new fields are added and existing ones are expanded upon or changed. This led to a need for generalizability in the interface, specifically in regards to searching and filtering. Two approaches were tried to address this issue. The first was a white list approach in which only certain fields were exposed and allowed to be filtered on. This ensured that the application would always exist in a working state, but required all new fields introduced to be appended to the list of filterable fields. In order to address this shortcoming, a blacklist approach was used to disallow filtering on non-keyword based fields and to allow all newly introduced fields to be default filterable, similar to Kibana. To take things a step further, the ability to directly enter an Elasticsearch query was also added and the interface was constructed in such a way to dynamically support displaying the results.

2) Downloading Files: The next issue the file observatory web application aimed to address was that of associating the metadata in Elasticsearch with the actual PDF files they represented. To do this, the AWS S3 object key associated with each PDF file was used as the custom ID for each document indexed in Elasticsearch. When a file or batch of files were then requested to be downloaded, these IDs were sent to the web application's API, where the referenced files were then pulled from S3 and sent back to the user.

3) Visualizations and Custom Aggregations: The last goal of the file observatory web application was to provide custom visualization and aggregation functionality that Kibana alone could not provide. This functionality took three forms.

The first custom functionality the web application added was for visualization purposes. Elasticsearch aggregation queries were automatically constructed for each filterable field and a breakdown of documents falling into each respective unique value for the field was used to populate a dynamic donut chart. This visualization works on top of the existing querying done in the interface and therefore serves as a representation of all visible search results.

The next custom tooling functionality that was added was the ability to see related misspellings of certain keys and a count of documents in which these misspellings were found. This was done using an Elasticsearch query similar to the one shown in Figure 4a. However, per the Elasticsearch specification, the associated document counts with these queries are approximations rather than exact numbers. Additionally, a document count for the actual search query was not provided. To remedy these problems, the misspellings and the search query are put into a list and a subsequent Elasticsearch aggregation query is fired to capture these exact counts and populate the table shown in Figure 3.

The last custom feature that was deemed relevant was slightly more complex. For a given key, we wanted to see all children keys that fell within its scope. Since this is naturally a nested attribute that would not easily be capturable by an Elasticsearch query, the key hierarchy structure was flattened and added as a new field to each document. Then, a query such as the one shown in Figure 4b was used to overload Elasticsearch's prefix completion capability to provide a list of all children keys. The result of this custom query was then output in a table as shown in Figure 2.

C. Applying Machine Learning

The file observatory dataset is undoubtedly dense, and therefore one of the crucial elements of understanding it is

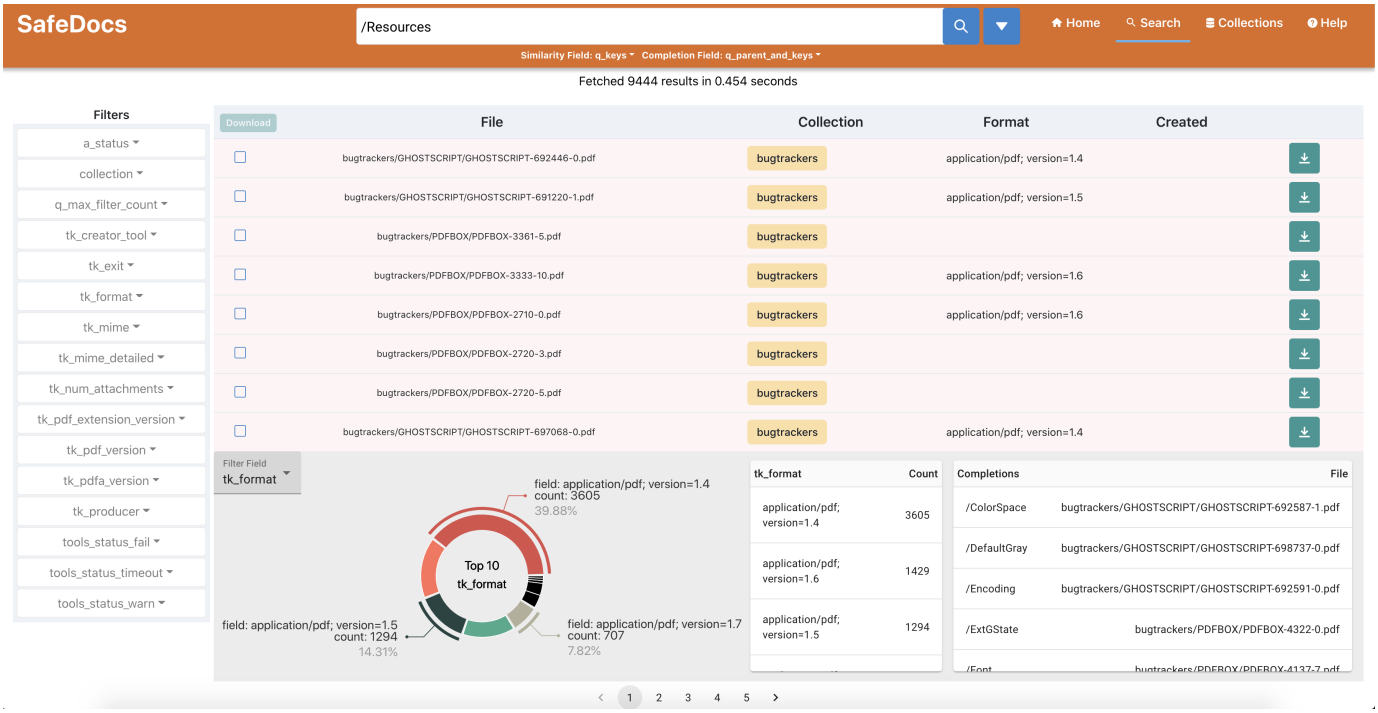

Fig. 2: Web application for the file observatory demonstrating the capabilities of searching, filtering, dynamically visualizing data, and seeing children keys.

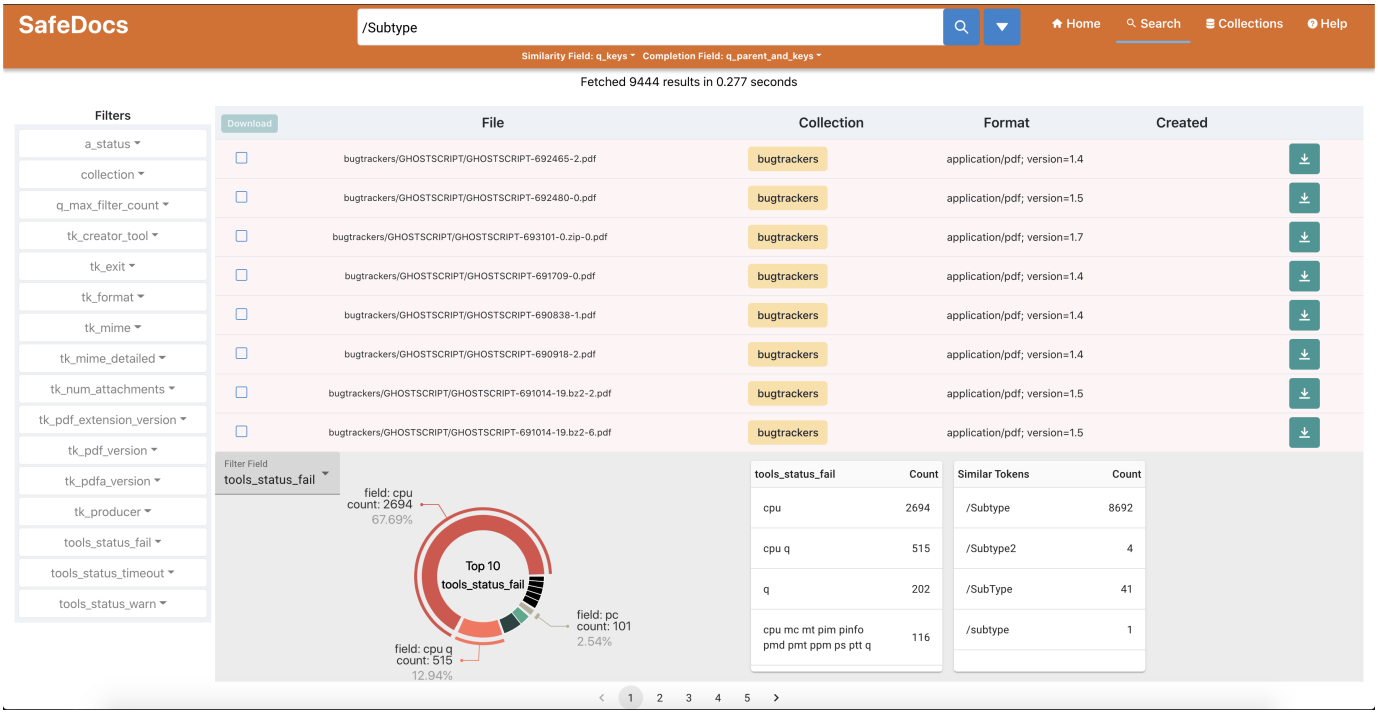

Fig. 3: Web application for the file observatory demonstrating the capability of automatically showing related misspellings that are within a Levenshtein edit distance of 2.

exploring the data with the help of visualization and machine learning. For the sake of simplicity, only a portion of the dataset was used in the following operations. The first step of the dataset exploration involved analyzing the creator tools used for each PDF document and their correlation to the PDF versions by calculating the chi-square and consequently the

```
GET file-observatory-202012/_search
{
  "_source": false,
  "suggest": {
    "my-suggest1": {
      "text": "/Subtype",
      "term": {
        "field": "q_keys",
        "suggest_mode":"always",
        "sort":"frequency",
        "size":100,
        "max_edits":2,
        "min_word_length":2,
        "max_term_freq":2000000
      }
    }
  }
}
```

(a) Misspellings query using the Levenshtein edit distance.

```
GET file-observatory-202012/_search
{
  "_source": false,
  "suggest": {
    "YOUR_SUGGESTION": {
      "prefix": "/Resources",
      "completion": {
        "field": "q_parent_and_keys.completion",
        "size": 2000,
        "skip_duplicates": true
      }
    }
  }
}
```

(b) Prefix completion query on a flattened key hierarchy field.

Fig. 4: Custom Elasticsearch queries that are used to extend functionality of the web application beyond that of Kibana.

p values. In short, the chi-square test allows to determine a statistical correlation between categorical variables in a dataset. That is, a lower p value (¡ 0.05) indicates a high chance of correlation between the PDF version and the creator tool used. The creator tools with the lowest p values can be observed in Figure 4.

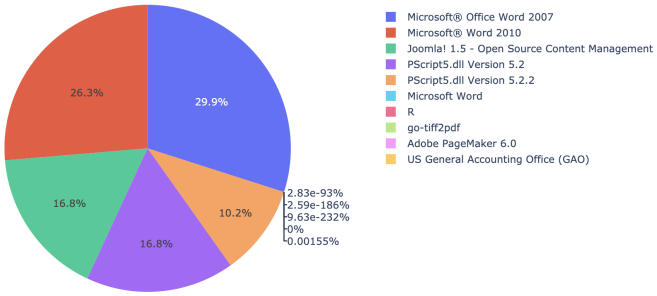

Fig. 5: The bottom 10 p values as per chi-square test performed on PDF creator tool vs PDF version.

Upon studying the possible importance of the creator tools in the file observatory dataset, it became apparent that further exploration of the tools in relation to other variables within the dataset could yield fruitful results and improve the understanding of the trends within the documents. It was therefore decided to experiment with various classification methods to test the possibility of predicting the creator tool of a document given a list of PF keys attached to it.

Each document comes with a multitude of variables, but one of the most important ones is the list of polyfile (PF) keys. Although each document comes with its own unique list of PF keys attached, a lot of them are quite frequent among most documents. However, due to the volume and complexity of said keys, simply exploring the data did not seem very promising. Therefore, classification was performed on the files, where the creator tool of each file was treated as a label, and the PF keys were treated as features. Due to the fact that the dataset contains over 300 unique creator tools, the top ten most occurring ones were selected in order to simplify the problem, which resulted in a final dataset of 1242 elements, where the label was one of the top ten creator tools. It is important to note that the PF keys listed for each document were retrieved in alphabetical order, so the order was not a determining factor in the classification process. However, upon early examinations, experiments with synthetically generated values, where the order of the keys was randomly shuffled, were performed in order to test the importance of the size of the dataset in the classification process. It was concluded that synthetic variables created to improve the volume and the variety of the dataset only hindered the performance of classifiers when applied to the dataset.

TABLE I: Classification Results on the File Observatory Dataset

Classification Method	Train Accuracy	Test Accuracy
Multinomial NB	82%	80%
Gaussian NB	62%	60%
Bernoulli NB	69%	66%
Logistic Regression	94%	90%
SVM	91%	88%
Decision Tree	N/A	89%

To analyze the performance of each of the classification methods, the final test and train scores were calculated. For some of the classifiers, the feature importance was also analyzed. The data was first transformed using the count vectorizer. The classifiers tested included multinomial naive bayes, gaussian naive bayes, bernoulli naive bayes, logistic regression, SVM, and a simple decision tree. Although the three naive bayes algorithms underperformed (see Table 1), the latter three algorithms performed quite well, with logistic regression taking the lead. The feature importance scores were also calculated and normalized using the min-max rescaling method. Upon exploring feature importance within some of the classification methods applied to this dataset, it was apparent that all of them differ quite drastically per each label and per classifier. However, some similarities and trends can be spotted. For an example of top ten most important features see Figure 5.

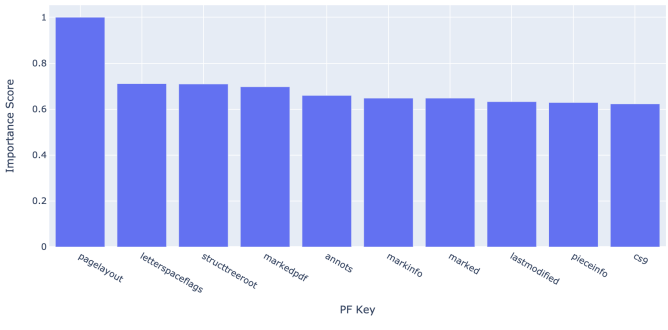

Fig. 6: Top 10 important features for the *Acrobat PDFMaker for Word* class using logistic regression.

Although applying various classification and visualization methods on the file observatory dataset has certainly helped with exploring and understanding the data in more depth,

and some good results were yielded, further improvements to the methodology and analysis can be made, and the resulting classifiers could be trained and tested on a larger portion of the dataset.

V. Future Work

We look forward to continuing to scale our file observatory, making it publicly available, and making it more generalizable for both the feature-engineering in the backend and, as we discussed, in the user interface.

Acknowledgments

This effort was supported in part by JPL, managed by the California Institute of Technology on behalf of NASA, and additionally in part by the DARPA Memex/XDATA/D3M/ASED/SafeDocs/LwLL/GCA programs and NSF award numbers ICER-1639753, PLR-1348450 and PLR-144562 funded a portion of the work. We acknowledge the XSEDE program and computing allocation provided by TACC on Maverick2 and Wrangler for contributing to this work. We would like to thank Peter Wyatt of the PDF Association and Dan Becker and his colleagues at Kudu Dynamics for their ongoing collegiality and collaboration on this task.

References

[1] T. Allison, W. Burke, V. Constantinou, E. Goh, C. Mattmann, A. Mensikova, R. S. Philip Southam, and V. Timmaraju, "Research report: Building a wide reach corpus for secure parser development." San Francisco, CA: LangSec (IEEE CS Security & Privacy Workshops), 2020.

[2] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. S2 – S11, 2009, the Proceedings of the Ninth Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287609000346

[3] "Apache Tika," https://tika.apache.org.

[4] "Apache PDFBox," https://pdfbox.apache.org.

[5] "Apache POI," https://poi.apache.org.

[6] "Apache Tika's Regression Corpus (TIKA-1302)," https://openpreservation.org/blog/2016/10/04/apache-tikas-regression-corpus-tika-1302, 2016.

[7] "Datasets for Cyber For'ensics)," https://datasets.fbreitinger.de/datasets/, 2019.

[8] S. Garfinkel, "Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus," *Digital Investigation*, vol. 9, pp. S80 – S89, 2012, the Proceedings of the Twelfth Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287612000278

[9] V. Basile, J. Bos, K. Evang, and N. Venhuizen, "Developing a large semantically annotated corpus," in *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012, pp. 3196–3200. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/pdf/534_Paper.pdf

[10] "Common Crawl," https://commoncrawl.org.

[11] P. Wyatt, "Stressful PDF corpus grows!" https://www.pdfa.org/stressful-pdf-corpus-grows/, November 2020.