Research Report: Progress on Building a File Observatory for Secure Parser Development

Tim Allison*, Wayne Burke†, Dustin Graf††, Chris Mattmann‡, Anastasija Mensikova§, Mike Milano¶, 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, ¶mike.milano@jpl.nasa.gov, ‖philip.southam@jpl.nasa.gov, **ryan.a.stonebraker@jpl.nasa.gov, ††dustin.j.graf@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 work reported at the LangSec 2021 conference on the development of a file observatory to gather and enable analysis on a diverse collection of files at scale. The initial focus of the observatory is on Portable Document Format (PDF) files and file formats typically embedded in PDFs. In this paper, we report on refactoring the ingest process, applying new analytic methods, and improving the User Interface.**

***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 enable file research for parser developers working on DARPA's Safe Documents (SafeDocs) program [2]. In our LangSec 2021 paper [3], we reported on improving our data collection by focused crawling of open source issue trackers and on improving our analytics capabilities by relying on advanced features of Elasticsearch. In this paper, we report on a major refactoring of our ingestion processes to facilitate technology transfer while maintaining scalability, new analytic methods to identify patterns in creator tool and parser failures, and in dramatic improvements to our User Interface. As with our earlier papers, 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.* led the way with GovDocs1 [4] 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 [5], Apache PDFBox [6] and Apache POI [7]) 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 [8] [9]. These corpora enable direct comparison of different approaches and tools under development, in addition to providing the means for reproducibility [4] [10] [11].

III. Refactoring Ingestion

In our [1] paper, we described a system that relied on cutting edge cloud components to scale data gathering, feature extraction and the ingestion of features into an Elasticsearch index to enable searching and advanced analytics. Our initial design made heavy use of Amazon Web Service (AWS)-specific tooling. We eventually settled on Amazon Batch to handle the scheduling, file processing and feature extraction. The parse results and features were stored in an Amazon Athena database, and then we had custom code to merge data across numerous tables into single feature documents per input document.

In demonstrating our capability to numerous stakeholders, we learned that tightly coupling our design to proprietary AWS tools created challenges for transferring this technology to others who may have access to only a limited set of AWS services or perhaps no access to AWS services at all.

In the last year, we pivoted to develop an "observatory in a box" that would lower the bar for adopting our ingestion components. The goal is to be able to run the main processes on a single machine or virtual machine (VM) and potentially distribute the CPU-intensive parts (file parsing and feature extraction) to as many VMs as are available.

As with our initial design, we have chosen to isolate processing resources (CPU and memory) and dependency management via Docker containers. We have begun wrapping parsers and post-parse feature extractors as Apache Tika parsers [12] in Tika's tika-server framework. This allows a single, commonly understood interface for all of the various parsers. Further, tika-server robustly handles timeouts and crashes, and it facilitates scaling across multiple VMs.

The current design allows for processing of files on a local hard drive or mounted fileshare, or, via the tika-pipes module, users may configure the ingest components to process files stored in AWS S3 buckets or Google Cloud Storage (GCS) devices. As mentioned, processing can happen on a single VM or on multiple VMs, and metadata and features can be stored on a local instance of Postgresql or on a cloud-hosted instance. This hybrid approach has been more appealing to potential transition partners.

While much work remains to wrap all of our parsers in the tika-server framework, we have made available examples on github [13].

IV. Data Sources with New Ingestion Framework

As we described in our earlier two papers, Common Crawl [14] offers a critical resource to gather enormous volumes of data for specific file formats easily. To support the development of our new ingestion pipeline, we rewrote our Common Crawl processing code to be run on a single virtual machine. We selected the July/August crawl from 2021, known as CC-MAIN-2021-31, as the target month for this exercise.

As noted earlier, Common Crawl truncates files at 1MB, and we have processing steps to politely refetch these truncated files from the original URLs. In the case of PDFs, we found that nearly a quarter of PDFs in CC-MAIN-2021-31 are truncated and require refetching.

We gathered 6.4 million non-truncated files from Common Crawl (1.6 TB) over the course of two days. We refetched 1.9 million truncated files (9.8 TB) over the course of two weeks. As noted, this refetcher was "polite", and we were careful to design it to throttle requests from the same domain.

V. Data Analysis

In our previous paper, we showed how using features built into Elasticsearch allowed for identifying structural variants and hierarchical anomalies in the data. In the analysis this year, we focused on parser outcome analytics. The goal was to identify patterns and relationships between creator tools (e.g. Adobe, Microsoft or smaller projects) and parser outcomes (e.g. qpdf, Apache Tika, poppler). We distinguished three outcomes: 1) 'success' – the parser completed within a reasonable amount of time without warning messages, 2) 'warn' – the parser completed successfully but wrote warnings to stdout or stderr, 3) 'crash' – the parser exited with a non-zero exit value or took longer than reasonable. We heuristically set timeout limits per parser because of such different "typical" behavior for each parser. When referring to outcomes per creator tool in this context, we are talking about the outcomes of parsers (e.g. caradoc, Apache Tika) when processing documents created by specific creator tools (e.g. Acrobat Distiller).

For this analysis, we focused on a SafeDocs-internal dataset of roughly one million documents ("eval three"). This dataset largely derives from Common Crawl, but it also includes synthetic fuzzed files. For each file, we gathered outcome status across a wide range of parsers. In the following, we used a subset of 30,000 files for demonstration purposes, and we focused on outcomes on files created by the more common creator tools.

In previous work, we used machine learning to explore this type of data due to its ability to capture complex patterns. This again was our initial approach, but traditional data analytics and statistical techniques proved both sufficient and more performant to determine trends within the outcome data for this use case. In future work, we will investigate ways in which to make this analysis more efficient to allow for integration into the User Interface so that it may work on more dynamic data.

One of the key statistics that we wanted to explore was the percentage of crashes for each creator tool, where the following formula was used – total number of parser tool crashes per each creator tool divided by the number of files created by that creator tool. For example, if *Acrobat Distiller Windows* was the creator tool for 10 documents and *caradoc* (*cd*) crashed for 2 of those documents, then *caradoc*'s crash rate on files created by *Acrobat Distiller Windows* is 2/10, or 20%. When observing real data in Figure 1, we can see that *caradoc* (*cd*) crashed 99% of the times *Acrobat Distiller Windows* was used to generate a PDF document. Analytics and visualizations like this were created for other commonly occurring creator tools, and it was noted that parser tools such as *clamav* (*c*), *cd*, and *pdfcpu* (*cpu*) crashed more frequently than other parsers for documents in our sample.

Another method of statistical analysis that was used for studying the data was the Chi-Square test. The Chi-Square test is generally used to determine whether or not certain data points differ from their "expected" values, or in other words, if they fall out of normal ranges and do not follow the general trends of the dataset. In our case, the Chi-Square test determines if there is an association between parser outcomes and creator tools (that is, whether they are independent or likely to be related). To perform this test, the number of observed (O; the total number of times each parser tool crashed for each creator tool in our sample dataset) and expected (E; the total number of crashes for each parser tool in the entire dataset divided by its size and multiplied by the total number of documents created by the given creator tool) values were calculated using the formula below.

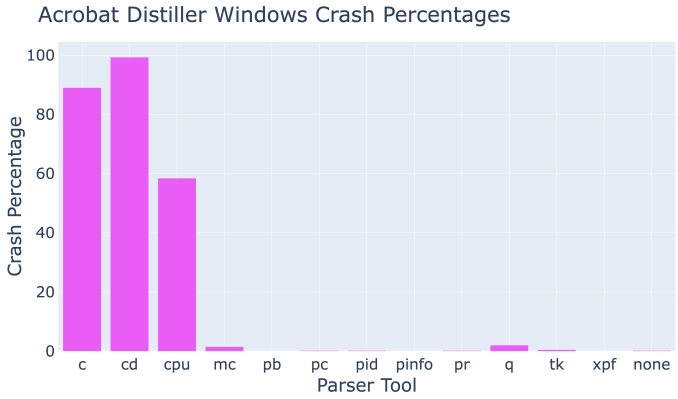

Fig. 1: Parser crashes on Acrobat Distiller Windows-created documents.

$$\chi_c^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

	Letter	Creator Tool
0	A	Acrobat Distiller Windows
1	B	TCPDF PHP
2	C	SelectPdf Html To Pdf for NET Community Edition v
3	D	TCPDF
4	E	Mac OS X Quartz PDFContext
5	F	Adobe PDF Library
6	G	Acrobat PDFWriter for Windows
7	H	Bullzip PDF Printer Freeware Edition
8	I	mPDF
9	J	iText by lowagiecom

Fig. 2: A legend for creator tool codes in Figures 3 through 6.

The p-value for each parser tool was also calculated. The p-value aids us in understanding whether or not we can accept or reject the null hypothesis, which in this case, told us if the variable was independent or not. The Chi-Square test was performed across the entire dataset of approximately 80,000 documents to obtain general trends within the data. However, the more interesting results and visualizations came out of testing just 10 creator tools used within that data, which ended up being a smaller dataset of 36,549 documents. Figure 3 shows the findings of this analysis, portraying significantly large Chi-Square values for *Acrobat Distiller Windows*, *Adobe PDF Library*, and *Mac OS X Quartz PDF Context*, and therefore showing how statistically different those values are from expected values (a legend for creator tool encodings in 3, 4, 5, and 6 can be found in 2). For a closer look at Chi-

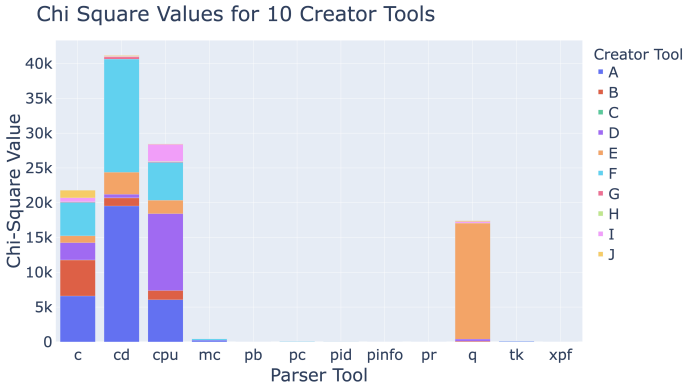

Fig. 3: A stacked bar chart portraying chi-square values based on parser tool crashes for 10 creator tools. A small chi-square value indicates a high correlation between the observed and expected values, meaning that the number of crashes was about the same as expected, and vice-versa.

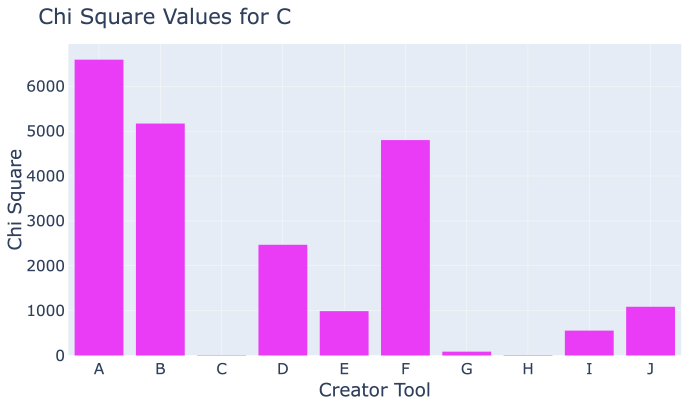

Fig. 4: A bar chart portraying chi-square values for clamav (c) and 10 creator tools.

Square values of the parser tool *c* see Figure 4, where we can once again observe the statistical difference its outcome on files created by *Acrobat Distiller Windows* and *Adobe PDF Library*. Similar visualizations were built for the remaining parser tools. However, it is important to note that given the smaller size of this dataset, a lot of parser tools were successful (that is, did not crash) for a lot of creator tools. A heat map portraying the magnitude of Chi-Square or p-value as color in two dimensions was deemed as the best way to portray these findings. The general heat map of p-values for this data, as observed in Figure 5, and the heat map of Chi-Square values, as observed in Figure 6, showcase that a lot of the values are, in fact, dependent (or related) as there is no statistical difference between them. However, parser tools such as *c*, *cd*, and *cpu* are worth exploring further due to their observed and significant effect on parser crashes within the dataset, especially in conjunction with creator tools such as *Acrobat Distiller Windows* and *Adobe PDF Library*.

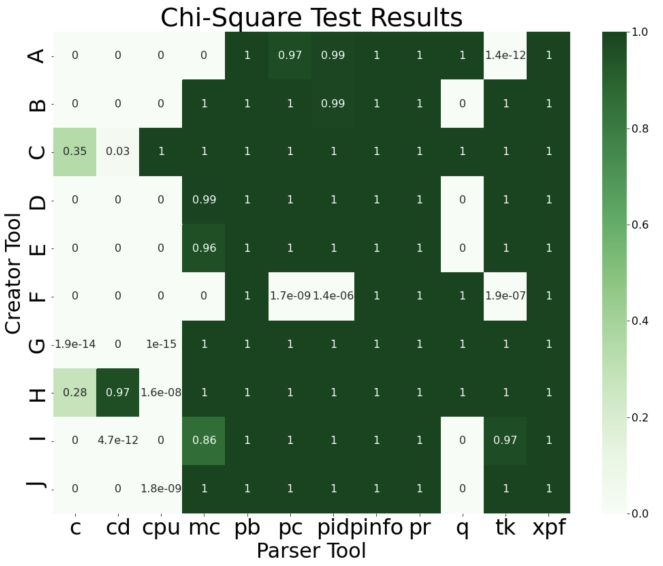

Fig. 5: A heat map portraying chi-square values for 10 creator tools.

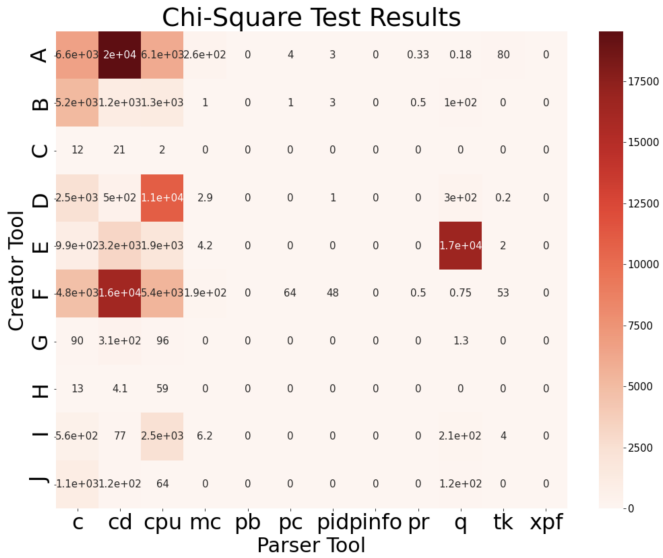

Fig. 6: A heat map portraying p-values for 10 creator tools. A small p-value indicates a high statistical significance.

VI. A New File Observatory Interface

In our LangSec 2021 paper [3] we introduced our first attempt at creating an interface for our File Observatory. However, as we further packed in new features and expanded the platform, overall usability started to decline. When designing the initial interface, there were several key assumptions made for how users would want to search for parser attributes. We thought that a universal search bar would align with a more traditional web application and best allow users to make more general queries for specific documents. We expected that users would conduct a broad search and then use the filters on the left side to further narrow down their query. Once a specific document subset was found, we then expected them to use our generic feature count graph, represented with a donut visualization, to gain insight into more specific parser attributes. This workflow worked well for some cases, but for many others, it seemed to involve an unnecessary amount of steps. Furthermore, it failed to address that in most cases, users would not be looking for specific documents, but rather subsets containing specific parser features (i.e. instead of having search functionality, everything should be tailored around filtering).

Additionally, as new features were introduced, both the information hierarchy and the use of screen space deteriorated and became cluttered. This resulted in an inability to intuitively understand what to focus on, where to start querying, and inconsistencies in features being surfaced depending on what was queried. In order to address these issues, we decided to first conduct a full-scale heuristic evaluation of the web application to figure out what was working, what needed to be rethought out, and most importantly, how users perceived the tool as a whole.

A. Heuristic Design Evaluation

A heuristic evaluation is defined by the Nielsen Norman Group [15] as:

> a usability engineering method for finding the usability problems in a user interface design so that they can be attended to as part of an iterative design process.

Conducting a heuristic evaluation was important not only from a design perspective, but also because it helped inform new feature requirements for the tool as a whole.

Through the heuristic evaluation, we were able to uncover several crucial usability issues. First, as touched upon earlier, the search bar landing page wasn't necessary. Most users use the application like a dashboard, and having a landing page just added an additional click. Furthermore, the default universal search bar itself proved to only occasionally be used, and did not warrant being the center of user attention. The toggleable, direct Elasticsearch querying capability of the search bar did seem to be beneficial as it introduced filtering that was not otherwise possible, but again, this was not used with every query. For this reason, we focused on making the search bar more minimal and increased the space given to the specific field filters that were more frequently interacted with. Once we addressed this though, it became apparent that the data visualizations were being given too little space as well given their usefulness and more could be done to highlight them. The last key features that needed to be addressed were the similarity and completion tables. These tables only conditionally appeared when a search term was entered and there were either similar values or values that completed the term found within the queried data. This proved to be confusing as a new user would miss out on their existence as a whole until they searched for something that populated them and did not clearly convey when there were no similar values or no completion terms found.

After finishing the heuristic evaluation, we decided to conduct more in-depth user interviews with Peter Wyatt and the PDF Association to ascertain a better understanding of how the tool was being used and what pain points could be alleviated from a user perspective. The key takeaways from these interviews proved validating of our heuristic evaluation and further emphasized the need for a focus on the visualization capabilities our tool added. However, in talking to Peter Wyatt, it also became clear that Kibana, a tool built specifically for exploring Elasticsearch data, still had a place in the observatory workflow. This was a key insight because it directed our attention away from recreating Kibana's existing functionality and instead focusing more on how we could extend it.

B. Designing a New System

After conducting thorough evaluations of our existing system, we began creating a new design for the next iteration of the application and re-thinking how we could best solve the problems that existed in the previous version. One of the big shifts with the new application was the decision to shift from a web-based system to a desktop application. By building the Observatory as a desktop application, we were able to extend functionality beyond the limitations of a webpage and lay groundwork for creating a more generalized system. In order to avoid previous pitfalls while designing out the UI, we first created design simulations to further test how users would interact the tool. A design simulation is performed by taking the mock-ups and adding in simplistic interactions to show how the application is supposed to work and emphasizing where screen transitions occur. This was crucial because it allowed us to validate actual use of the application before beginning development.

In designing a new interface, we increased focus on both modularity and scalability. The reason for modularity is that we wanted the Observatory to be able to have its underlying data source changed without the interface having to drastically update as well. Modularity also provides the foundation for scalability, which further allowed us to explore how best to integrate a dynamic and growing tool set.

The end result of our usability and design studies was an application that had a more succinct information hierarchy, less cluttered and wasted space, more immediate and directed interactions, and more room to expand. We were able to implement multiple new features as well such as more generic visualization graph types (donut, bar chart, and treemap), a visualization of field-specific significant terms, a geospatial mapping of where documents are hosted, and two third party integrations with Polyfile's Hex Editor (referenced later) and an integrated portal to Kibana (Figure 7). Another small but impactful change was utilizing a mono-space font to allow for fast and accurate comparison of differences in long strings of text. Other key changes were the introduction of the settings and mapping pages, which allow for users to customize what parser attributes they want visible in the search view table, what fields they want to be able to visualize, and what fields they want to be able to filter on. In total, these changes allow the user to enter a query from multiple starting points and refine that query without having to backtrack and instead focus on being able to thoroughly investigate parser attributes.

C. Generalizing the Backend

In creating a revamped interface, there were several points of consideration. As we described from a design perspective, there were many things that needed to change in order to enable more dynamic content, visualizations, and set things up for success with future integrations. From a development perspective though, this also meant the removal of many previous assumptions. We could no longer assume that certain columns would exist for visualization purposes, nor could we assume at all what types of columns would be in a generic data collection. This meant that a huge emphasis of the new interface development had to be customizability and error handling; allowing the user to select columns and configure settings for use with all of the different tools and visualizations we wanted to include and not crashing if they selected incompatible ones.

As mentioned previously, Kibana handles many of these issues off the get-go and was a common tool paired with our system. However, the key benefit of our old UI was that it extended these capabilities and allowed dynamic, real-time visualizations to be created that required multiple queries and more complex logic than the Elasticsearch querying language could directly provide. In rebuilding the interface, we decided to lean further into this area and separate the application out from the confines of a web-based application to an Electron desktop application. Electron is a node.js framework that uses a Chrome driver to generate cross-platform compatible desktop applications that can be developed using a traditional web stack. The benefit of this was that we could port parts of our previous UI over to reduce new development time and also now have the added ability to better interact with the user's file system and run command line tools. Additionally, this also meant we could go a step further and directly integrate Kibana as a third party tool, allowing us to take full advantage of its power as an exploratory tool.

Another key area of development we needed to focus on was the granularity of customizability we wanted to expose. Going forward to a generalized, locally-running file observatory, we no longer wanted to limit users to only being able to view data on a single index in our remote Elasticsearch instance. We instead wanted to provide users full control to dynamically point the tool at any index and any base instance of their choosing. This meant providing support for three modes of connection; connecting to a locally running Elasticsearch instance, connecting to a publicly available remote instance, or in the case of our hosted, remote data store, connecting to an API that acted as a passthrough to Elasticsearch. To add to this complexity, the use case also existed where there could be a locally running, sampled mirror of an index that also existed remotely on another Elasticsearch instance. In order to preserve configuration settings in this case, while

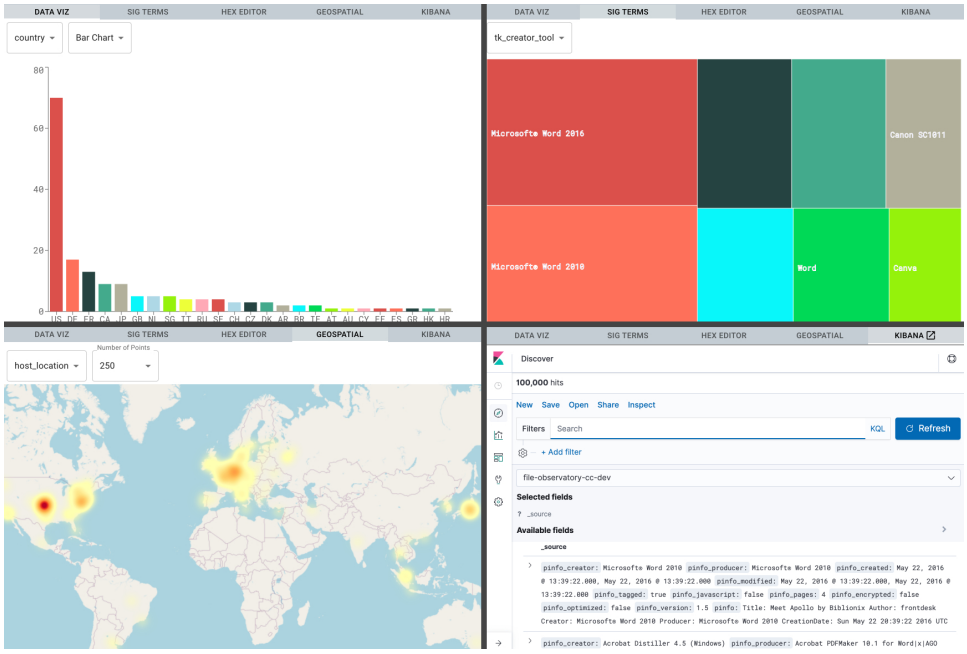

Fig. 7: The generic data count visualization, significant terms visualization, geospatial visualization, and Kibana integration tabs in the User Interface.

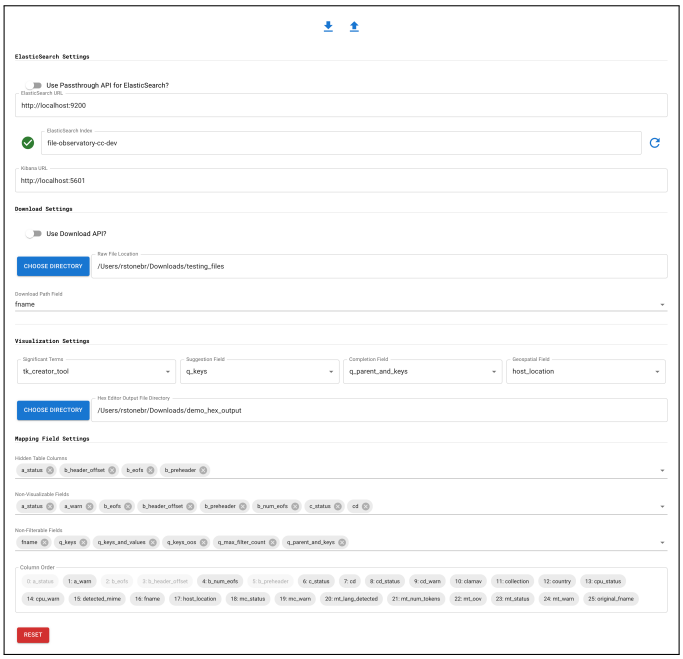

Fig. 8: Settings screen in the UI that is used to generate and load a JSON-based configuration file. It allows for full configuration of Elasticsearch and file system connection details, which fields to correlate with specific visualizations, and provides control over where specific fields appear and where they are hidden.

also allowing unique settings to be configured for different indices, we came up with a JSON-based configuration format that is keyed by the Elasticsearch index and has connection information stored as additional properties of that index. This allows for the same index to be connected to multiple different base Elasticsearch instances while still maintaining all of the subsequent configuration details for it. To further enhance the portability of this feature, we also introduced the ability to import and export the configuration file. All exposed configuration options we introduced are shown in Figure 8.

D. Integrating Third Party Tools

As previously mentioned, one of our key goals with our new observatory was to expand beyond what was capable of an in-browser web application. One area that benefited from this focus was our ability to more seamlessly integrate with external tools that were developed as part of or used for the DARPA SafeDocs program. While much of this potential use case still remains unexplored and will be an exercise for future work, the first release of our new observatory app focused on integrating Trail of Bit's Polyfile tool [16], specifically its ability to generate a dynamic file structure breakdown and produce a bespoke, HTML-based, hex editor viewing tool as shown in Figure 9. Polyfile is a command line based tool that can be installed via Python's package manager, Pip. However, to directly use this would mean we would need to either make the assumption that the user already had Python and the tool installed and could provide us with the location of the executable or package both a Windows, Mac, and Linux standalone Python instance that we could use to install it. While either one of these solutions wouldn't be too difficult to implement to get just Polyfile running, it would introduce the possibility for different platform-specific behavior and would

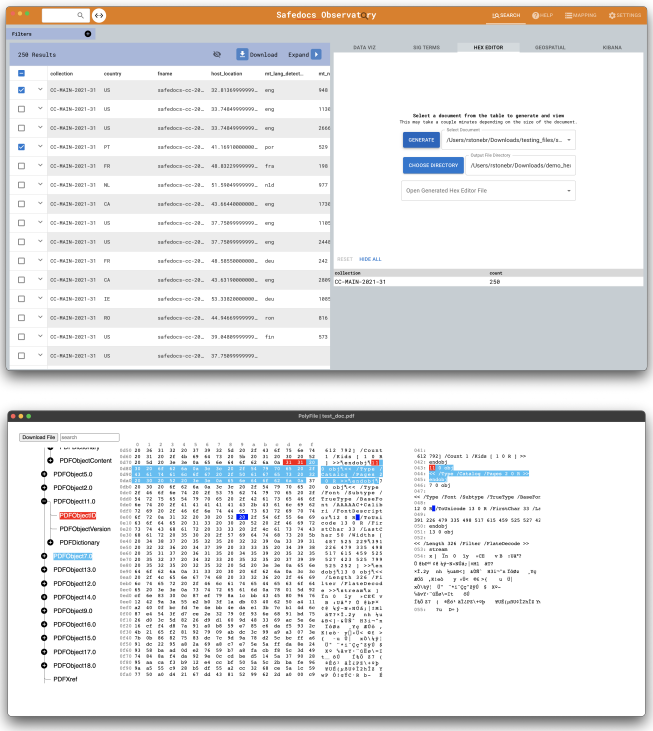

Fig. 9: The search view in in the User Interface with the Hex Editor tool tab opened and a dynamically generated Polyfile HTML Hex Editor document opened below it.

set a dangerous precedent for third party tool integrations in the future, especially if these tools were non-Python based and/or produced extra artifacts that our app would need to clean up. To address these causes of concern, we decided to instead use Docker.

The introduction of Docker into our toolchain solved many potential future problems off the bat. When a user first performs an action that would require output from a tool contained within a Dockerfile, our platform triggers a build of that Dockerfile. This gives us the option of either fixing the underlying tool to a specific version, or have it re-pull from it's respective package repository and always use the latest version. Once the Dockerfile is built, the command line tool is then used as an entrypoint and we have full access to use it from node. However, there still exists a disconnect between the target file and the tool as the target file is either stored locally on the user's computer or it is represented by an Elasticsearch document visible in the table view. In order to bridge this gap, a local temporary directory is mounted to the Dockerfile at run time. If a user selects a file from their computer for use with the tool, it will be automatically copied over to the temporary directory and the outputted HTML file will be moved back from the temporary directory to wherever the user set as the download location. If the user does not have the file on hand and instead selects it from the table view, a configured download path field will be used (shown in Figure 8) to locate the file, download it to the temporary directory, and send it to the tool without any additional interaction.

VII. Future Work

A. A Full-Scale Observatory in a Box

The introduction of a standalone, configurable Observatory application made great strides towards our planned goal of a self-contained, Observatory in a Box. However, one crucial area is still disconnected; the generation of the actual Elasticsearch dataset from raw files. Our efforts up to this point have separated these two parts as we focused on creating massive, cloud-hosted corpora and using the application simply as a means to view this information. However, now equipped with the ability to run Docker containers and integrate third party tools into the frontend, we plan on merging these two efforts to allow local Observatory indices to be fully generated from a single location given a document dump. While these local Observatories would not replace the much larger corpora we have hosted up to this point, it would provide a path towards transitioning our work in the DARPA SafeDocs program to other partners and entities who may have data that would benefit from analysis using this tool, but are unable to include it in our public datasets.

B. Expanding File Format Coverage and Handling Embedded Files

As we had planned from the beginning, we will focus on other primary file formats beyond PDF files. Relatedly, as part of the new effort in the front-end to allow for greater and easier configuration of fields, we have begun experimenting with using Apache Tika's embedded file capability. Specifically, Tika parses complex documents recursively, which means that it handles a wide range of file types even if the primary documents are PDF files. These embedded files may be user-added attachments or other files required to convey information for extraction or rendering (such as Adobe's Extensible Metadata Platform (XMP) data, font files, International Color Consortium (ICC) profiles and many others). In future work, we look forward to indexing more of the information about embedded documents and making the extracted features searchable and analyzable.

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, P. Southam, R. Stonebraker, and V. Timmaraju, "Research report: Building a wide reach corpus for secure parser development," in *2020 IEEE Security and Privacy Workshops (SPW)*, 2020, pp. 318–326.

[2] "DARPA's Safe Documents Program," https://www.darpa.mil/program/safe-documents, 2021.

[3] T. Allison, W. Burke, C. Mattmann, A. Mensikova, P. Southam, and R. Stonebraker, "Research report: Building a file observatory for secure parser development," in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 121–127.

[4] 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

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

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

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

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

[9] "Datasets for Cyber Forensics," https://datasets.fbreitinger.de/datasets/, 2019.

[10] 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

[11] 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

[12] "Apache Tika's JIRA," https://issues.apache.org/jira/projects/TIKA/summary.

[13] "File Observatory Tika Wrappers," https://github.com/tballison/file-observatory/tree/main/tika-containers, 2022.

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

[15] J. Nielsen, "How to conduct a heuristic evaluation," 1994. [Online]. Available: https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/

[16] C. Harmon, B. Larsen, and E. A. Sultanik, "Toward automated grammar extraction via semantic labeling of parser implementations," in *2020 IEEE Security and Privacy Workshops (SPW)*, 2020, pp. 276–283.