ExcavationPack: A Framework for Processing Data Dumps

TJ Nicholls

ExcavationPack:

A Framework for Processing Data Dumps

Author: TJ Nicholls, Packetvitality@gmail.com

GCTI Gold

Advisor: *Jonathan Risto*

Accepted: *March 8, 2021*

Abstract

Data dumped online from breaches is rich with information but can be challenging to process. The data is often unstructured and littered with different data types.
This research presents a framework using Docker containers to process unstructured data. The container-focused approach enables flexible data processing strategies, horizontal scaling of resources, the efficacy of processing strategies, and future growth. Security professionals utilizing this framework will be able to identify points of interest in data dumps.

1. Introduction

When online services are breached, the data is often posted online. Attackers frequently publish either samples or complete dumps of compromised data (Hunt, n.d.). Some or all of the information from a breach often becomes available on the public internet or underground internet marketplaces (Malwarebytes, n.d.). The type of data within a breach can vary widely. Common data types include personal data (name, address, email addresses, etc.), credentials (usernames, passwords), and financial data (credit cards, bank information, etc.). Figure 1 below summarizes the top attributes impacted by data breaches in 2020 (Verizon, 2020).

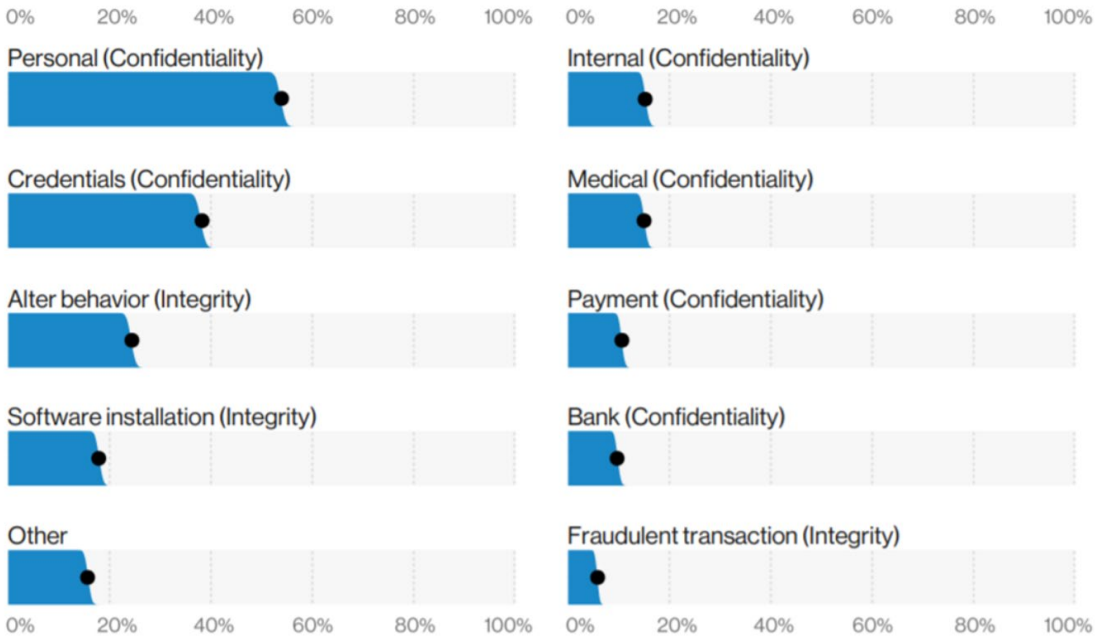

Figure 1: Top Attributes Impacted by Data Breaches

Data dumps can be offensive or defensive tools. One common offensive technique is credential stuffing. Credential stuffing can be defined as the automatic injection of breached username/password pairs to fraudulently obtain access to user accounts (OWASP, 2020). Similarly, defenders can use open-source intelligence (OSINT) techniques to locate interesting data in online data dumps and tests against applications, observe the results and recommend mitigations.

TJ Nicholls, Packetvitality@gmail.com

Data dumps may also impact an organization in ways that are not as obvious. In 2020, a data dump commonly referred to as 'BlueLeaks' was released and contained "ten years of data from over 200 police departments, fusion centers, and other law enforcement training and support resources" (Krebs, 2020). Additionally, "among the hundreds of thousands of documents are police and FBI reports, bulletins, guides and more" (Krebs, 2020). A review of the 'BlueLeaks' data dump revealed it also had copies of backend applications, including code repositories, which may typically not be available to threat actors. In these circumstances, the impacted parties must evaluate how the dump could be leveraged against them and implement mitigations.

2. Creating a Framework

There are no projects in the security community that enable users to filter for key data points within private data dumps. For example, https://haveibeenpwned.com will notify organizations or users if their data is involved in a breach, but the raw results are not available. Additionally, projects such as https://www.dehashed.com allow for searching raw results but present at least two critical problems. Firstly, the platform operators must have uploaded the data dump of interest and made it available for searching. Secondly, the platform operators have access to any sensitive data points that may be searched, requiring a level of trust.

The framework outlined in this paper uses a bring your data (BYOD) model to enable individuals and organizations to search data dumps privately. This research will detail how the framework attempts to fill this void in the community. The framework uses Docker containers to separate functionality, ease the burden of dependencies, and remove limitations of working within one programming language. The desired outcome is to assist the cybersecurity community in quickly identifying sensitive data in data dumps, enabling them to implement defensive measures.

2.1. Background Information

It may be helpful to understand the first attempt to solve the problem of filtering for key data points within a private data dump. The first iteration was using a single

Python script. The script had a function to categorize each file, a function for each data type, and other miscellaneous functions to handle specific scenarios. Once the script categorized a file, it used conditional statements to select which function should be used to process the file (one at a time). See Figure 2 below for a simplified illustration.

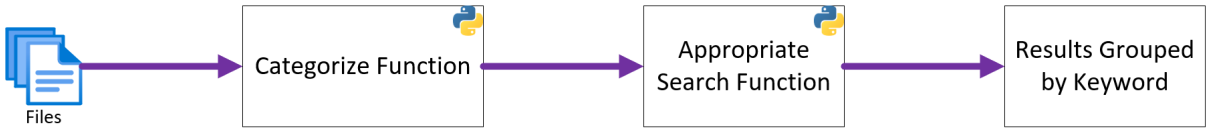

Figure 2: Simplified Initial Approach

The initial implementation began presenting several problems. For example, the script's utilization of the 'libmagic' file type identification library, used to categorize each file, produced inconsistencies across operating systems. A Python library called 'Python-Magic' provides an interface to the 'libmagic' library (PyPi, n.d.). To work on Microsoft Windows, a port of the native Linux 'libmagic' library to Windows is also needed. The script used the 'Python-Magic-bin' library for this purpose. In practice, using these libraries occasionally resulted in a file that was identified differently by the 'Python-Magic' depending on which operating system it was executed from. The details of testing file categorization in different scenarios are covered below.

2.1.1. Testing File Categorization

Figure 3 below contains the Python script used to determine the file type, perform a SHA1 hash of the file, and print the results.

```
def get_file_magic(filename):
    try:
        filetype = magic.from_file(filename)
        return filetype
    except Exception:
        return False
def hash_file(filename):
    try:
        h = hashlib.sha1()
        with open(filename,'rb') as file:
```

```
                chunk = 0
                while chunk != b'':
                    chunk = file.read(1024)
                    h.update(chunk)
            return h.hexdigest()
        except Exception:
            return False

filetype = get_file_magic(filename)
filehash = hash_file(filename)
print("Filehash=" + filehash + "\n" + "Filetype=" + filetype)
```

Figure 3: Python Script to Test File Categorization

Figure 4 below contains the results for testing the 'Python-Magic' library on Windows 10 using 'Python-Magic-bin'.

```
› python test.py
Filehash=f049cc6c6020d6452e760521febf17395a271c07
Filetype=block special
```

Figure 4: Windows Categorization Results with Python-Magic

Using the same Python script from Figure 3 above, the 'Python-Magic' library was tested on Ubuntu 20.04.1. 'Python-Magic' running on Ubuntu uses the 'libmagic' library natively installed on Ubuntu. Figure 5 below shows the test results.

```
› python test.py
Filehash=f049cc6c6020d6452e760521febf17395a271c07
Filetype=gzip compressed data…[truncated]
```

Figure 5: Ubuntu Categorization Results with Python-Magic

Lastly, Figure 6 below contains the test commands and the results for testing the 'file' utility running on Ubuntu 20.04.1. The 'file' also uses the native 'libmagic' library.

```
$ file filename
211sfbay.tar.gz: gzip compressed data…[truncated]
$ sha1sum filename
f049cc6c6020d6452e760521febf17395a271c07  filename
```

Figure 6: Ubuntu Categorization Results with 'file'

2.1.2. Discussion of Findings

The testing showed that file detection remained consistent while using the 'file' utility and the 'Python-Magic' library running on Ubuntu 20.04.1, but the 'Python-Magic' (using 'Python-Magic-bin') library running on Windows 10 provided different results. One logical conclusion of this behavior is that the port of 'libmagic' to Windows via the 'Python-Magic-bin' differs from the native Ubuntu library. These findings were an indication that ported libraries may not always be reliable.

2.1.3. Exploring New Strategies

Based on research of how others in the community may have approached similar challenges, the 'fulltext' open-source library notably had a very similar goal. Fulltext extracts texts from various document formats. It can be used as the first part of search indexing, document analysis, etc. (Timby, 2018). Figure 7 below shows a partial view of the data types supported by Fulltext and the tool or library used to process the data type.

Supported formats

Extension	Linux	Windows
`bin`	python stdlib	python stdlib
`bmp`	tesseract CLI and pytesserac module	
`csv`	python `csv` module	python `csv` module
`doc`	`antiword` CLI tool	
`docx`	`docx2txt` module	`docx2txt` module
`eml`	`email` module	`email` module
`epub`	`ebooklib` module	`ebooklib` module
`gif`	tesseract CLI and pytesserac module	
`gz`	python `gzip` module	python `gzip` module
`html`	`BeautifulSoup` module	`BeautifulSoup` module
`hwp`	`pyhwp` module as CLI tool	
`jpg`	tesseract CLI and pytesserac module	
`json`	`json` module	`json` module

Figure 7: Example of Formats Supported by Fulltext

In the case of Fulltext, the limitations of processing specific data types generally lie with the Windows operating system. However, this is not guaranteed, and there may be a preferred Windows tool or library which is not supported on other platforms.

It was apparent from the inconsistent results with the 'libmagic' library and the operating system dependencies for specific tools that a more flexible approach is needed. Processing may require the use of more than one programming language and operating system. The ideal approach is to evaluate each use case needed for the framework, determine the best tool, and build a solution using the best tool. It is also advantageous to use existing tools and libraries rather than writing a new one if it does not exist in a specific language or operating system.

2.2. Conceptualizing a Solution

The framework is focused on identifying points of interest within a data dump containing an unforeseen number of directories, subdirectories, and files. Files stored in an unstructured tree are common, as evidenced in 'Cit0Day' breach collection (Hunt, 2020) and the 'BlueLeaks' data dump (Krebs, 2020). The strategy taken to solve this problem can be summarized into two simple steps:

1. Categorize each file into data types.
   **Examples of data types include plaintext, Excel spreadsheet, PDF, etc.*
2. Select the appropriate 'workflow' to process the data type.

2.3. Core Design Principles

While designing the framework, the following core principles drove the decisions: Ease of Use, Programming Language Agnostic, Accurately and Reliably Locating Data, Scalability, and Modularity. The core principles are in no particular order.

2.3.1. Ease of Use

Security professionals may not have cycles in their workday to approach the data processing challenge of searching data dumps. Users of this framework should indicate the data they want to search, what interests them, and begin processing. Technical dependencies should be obfuscated as much as possible.

2.3.2. Programming Language Agnostic

There are many prebuilt libraries and tools available for parsing various data types. For example, Python may have an established library for parsing Excel files while

it lacks a library for PDFs. Another language or tool may offer better options for parsing PDFs. The framework should utilize the best available means, regardless of which programming language it was written in or which operating system it is dependent on.

2.3.3. Accurately and Reliably Locating Data

Security professionals would presumably be using such a framework to locate information of some level of importance. Failing to find the data and reporting false negative(s) would provide a false sense of security and could cause damages to the user and/or the organization. There should be a measurable way to test efficacy.

2.3.4. Scalability

Data dumps can be quite large. For example, 'Blueleaks' was approximately 270 gigabytes (Krebs, 2020). The framework should provide mechanisms where the user can add resources to reduce the total processing time. Security professionals may want to search through a data dump multiple times as new information becomes available. Waiting for long periods of time for each run can be discouraging, and in some cases, locating information is time-sensitive.

2.3.5. Modularity

There are many different data types, and it is challenging to forecast everything that could appear in a data dump. The framework should be modular so that support for data types can be added, removed, and improved over time. Furthermore, it should be structured in such a way that others in the community can contribute to the project if desired.

3. Technical Implementation

The application development has to account for many considerations to implement the core design principles. Examples of new application considerations include: allowing multiple 'functions' to run concurrently, enabling a diverse set of tools to be used, and not requiring the user to install a long list of dependencies. The solution to these challenges is explained in greater detail in the sections below.

3.1. Docker Containers

The functions were broken out of a monolithic script and ran inside containers to address lessons learned in the initial implementation and follow the core design principles. For example, a container using the native Linux 'libmagic' library running in the minimal Debian-based 'python:3.8-slim-buster' image (Docker, n.d.) took the role of categorizing each file. The same concept holds for each 'function' used for a particular data type. See Figure 8 below for a simplified illustration.

Figure 8: Substituting Containers for Functions

Running each 'function' in a container is similar to the idea of writing traditional programming functions. A function can be described as “a block of organized, reusable code that is used to perform a single, related action" (Tutorialspoint, n.d.). Containers take it a step further by reducing the burden of managing dependencies since they can include software dependencies needed by the application, such as specific programming language runtimes and other software libraries (Google, n.d.).

Running each function in a container also provides the flexibility of utilizing whichever programming language, tool, and/or operating system is best suited for the task. The variety of options available is evidenced by more than 150 official container images currently hosted in the repository Docker Hub (Docker, n.d.). Lastly, it may be desirable to host the framework on a server. Designing from the ground up with containers would allow the framework to run reliably from one computing environment to another (Docker, n.d.).

3.2. Maintaining State

Designing the framework to utilize separate containers provides many advantages, such as the ones listed above, but it also introduces complexities and additional application requirements. Many of these considerations are related to the fact that

containers are ephemeral (Benevides, 2016). Below are a few of the concerns that must be accounted for to enable the containers to run independently:

- A container must identify which data types it is responsible for processing, rather than the function/container being selected as part of a conditional statement like in the initial script.
- A container must be able to identify when additional processing is needed or if all processing is complete. For example, there may be files still waiting to be categorized while all of the current files have been categorized and processed.
- For the containers to run concurrently, they need to be aware of the current state of processing. For example, in addition to avoiding already processed files, a container should avoid files currently being processed.

The added complexities provide advantages to the processing workflow. Namely, the separate containers can all begin working at once, and the framework can initiate multiple instances of each container. The new capabilities mean that all file types begin processing as soon as they are categorized. It also provides the ability to add containers of the same type to increase processing power. In other words, if there five 'Plaintext Searcher' containers initiated, then the framework can process five text files concurrently.

Figure 9 below illustrates multiple containers working in tandem. Also, note that each type of container is utilizing a different programming/scripting language. For example, the Categorizers and Excel Searchers utilize Python, the Plaintext searchers utilize Go, and the Pdf Searchers utilize Bash. Only two instances of each container are illustrated below; however, the quantity of containers is controlled by the user.

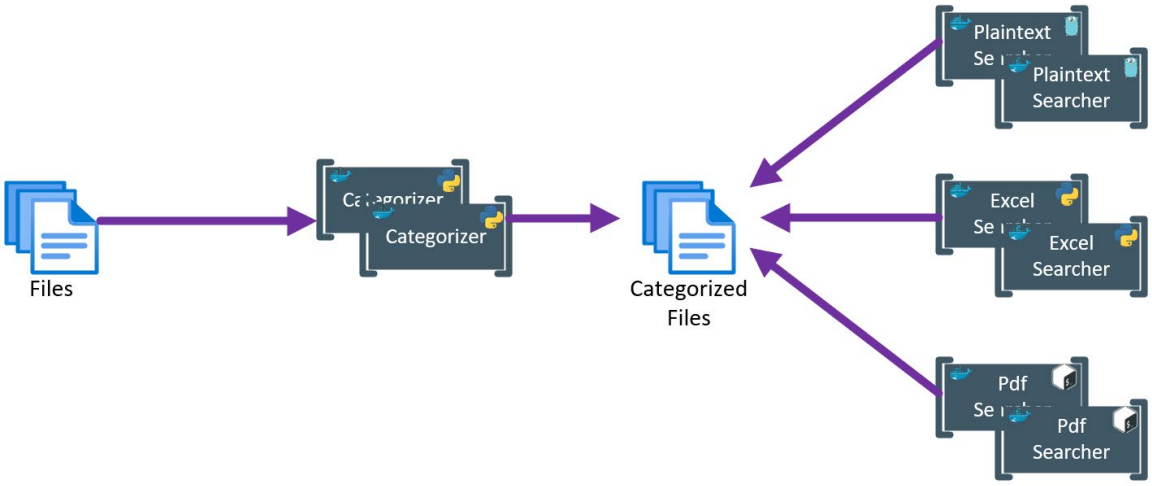

Figure 9: Multiple Containers working in tandem

3.3. Database

A SQLite3 database is used to store the persistent data for the various containers. SQLite3 was chosen for a few reasons. It is a relational database that is widely supported and easy to set up (Jagtap, 2019). Another crucial reason was that SQLite3 allows multiple applications to read data simultaneously and provides a means for multiple applications to write to the database when using write-ahead logging (WAL). SQLite3 will allow multiple applications to have the database file open at once and multiple processes to read the database. If an application wants to write, it must first lock the entire database file for the duration of its update (typically a few milliseconds) (SQLite, n.d.). Simultaneous writes are queued up in a WAL file and committed to the database when possible (SQLite, n.d.).

Tables were built to address specific considerations such as those outlined in Section 3.2 “Maintaining State,” above. SQLite3 is a relational database that makes it easy to establish relationships across different tables using a key (Oracle, n.d.). The database tables enable the separate containers to identify the current state of processing, attempt to avoid redundant processing of the same file, and rapidly update to the current state of processing.

The appropriate tables in the database have a relationship with one another by enforcing a foreign key restraint, utilizing SHA1 file hashes as the foreign key. Foreign keys constraints are a construct of SQL and are used to enforce "exists" relationships between tables (SQLite, n.d.), which in turn enforces referential integrity (Hayes, 2011). For example, there is a 'state' table to track where a file is in its processing lifecycle. A successfully processed file will move in the following order: Categorizing → Categorized → Processing → Processed. There is also a 'categorization' table that identifies the data type of a file. Maintaining state enables containers to enumerate files that are available for processing quickly.

Utilizing the database, containers supporting each function of the framework and multiple containers of the same type can process files concurrently. The containers monitor a general 'status' table and either wait for new data to be categorized or exit. See Figure 10 for an illustration of how the containers and database integrate.

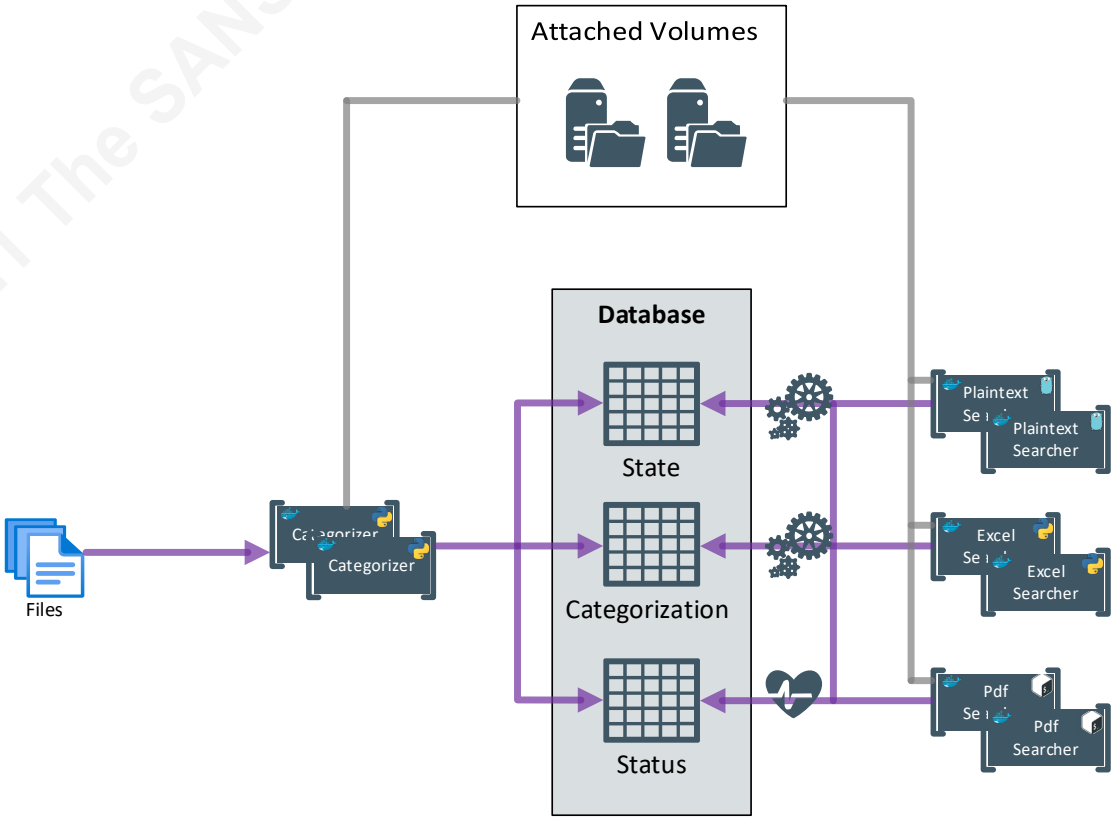

Figure 10: Container and Database Integration

3.4. Container Orchestration

Complexity is added to the initial processing when containers have different build requirements, types, and dependencies. To keep true to the core design principle of being easy to use, a single command using a docker-compose file can create and start all the services (Docker, n.d.).

Figure 11 below depicts the relationship of docker-compose to the containers in addition to a dependency between the containers. In the context of dependencies for docker-compose, one container which depends on another will not wait until the other container is "ready" (whatever that means for the application). It only waits until it is running (Docker, n.d.). The dependency provides a more substantial likelihood that the database will be initiated by the 'categorizer'. The 'searchers' are programmed to try and connect to the database. Upon failure, the 'searchers' will wait before trying again.

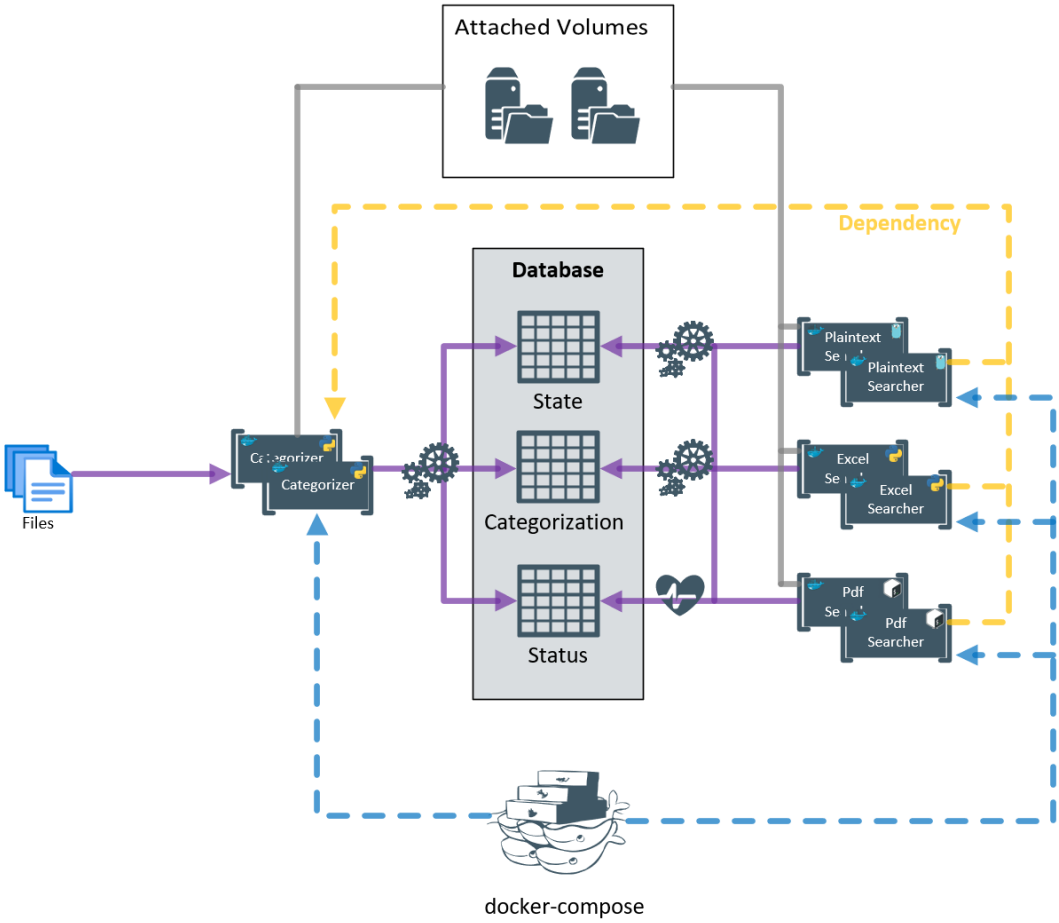

Figure 11: Docker Relationships

The docker-compose file provides other advantages as well. For example, a 'replicas' configuration setting allows the framework to easily spin up multiple instances of the same container. Another useful function is the ability to use a file to set environmental variables. The docker-compose file can help when multiple 'services' or containers within the docker-compose file have a similar setting. A variable can be set within the environmental file and referenced within the docker-compose file (Docker, n.d.). Overall, docker-compose has many features to help manage a multi-container setup and obscures complexities away from the user.

4. Identifying the Best Processing Strategy

Each component of the framework represents a preferred strategy for processing a data type or performing a specific function. The framework has adopted a Champion/Challenger approach to compare competing strategies. This approach enables users of the framework to observe multiple solution streams to reach an optimal decision. A Champion/Challenger approach compares two or more strategies to promote the one that performs the best (Berlioz, 2014).

The Champion/Challenger approach couples well with the container-based design of the framework. Containers offer a logical packaging mechanism (Google, n.d.) and are self-sufficient applications (Nishanil, 2018). When a Champion is determined, a new container can swap out the existing one. The framework is set up to enable improvements as new information becomes available and encourages experimentation.

4.1. Determining a Champion

A Champion is determined by observing which strategy is most successful at the designated task. Success is determined by observing which strategy best satisfies the pre-defined key performance indicators (KPIs). KPIs can be defined as the critical indicators of progress toward an intended result. They provide a focus for improvement and create an analytical basis for decision-making (KPI, n.d.). The Champion will be determined by a composite score comprised of reliability and comprehensiveness using the metrics below:

- (SP) Successful Parsing. For example, how many files were successfully processed vs. those that were left unprocessed or in an error state? This KPI is measured based on percentage.
- (TP) True-positive hits. In other words, how many times the strategy successfully identified a data point. This KPI is measured based on percentage.
- (FP) False-positive hits. In other words, how many times the strategy falsely identified a data point. This KPI is measured based on percentage.
- (T) Average processing time for all files. Measurements will ignore files ending in an error state. This KPI is measured with a duration.

The KPIs are used in the formula $\boldsymbol{SP(TP - FP)}$ to determine a Champion. The result of the formula is a composite score that prioritizes successful and accurate processing of data. In a tie, the strategy with a lower average processing time is considered the winner.

5. Using the Framework

The framework works wherever Docker and docker-compose are supported. It is initiated using a single docker-compose file accompanied by an environmental variables file. The framework can be found at https://github.com/packetvitality/ExcavationPack. The Github repository is the best reference for up-to-date guidance for using the framework. To summarize, the framework can be used with the following steps:

1. Ensure Docker and docker-compose are operational.
2. Edit the environmental variable (.env) file to specify a working directory and the directory to be searched.
3. Place a keyword text file in the working directory.
4. [Optional] Customize the docker-compose file.
5. Issue the command ‘docker-compose up’ from the project directory.

Customizing the docker-compose file provides more processing control. For example, users can configure the 'replicas' setting to create more or fewer instances of a specific container. The user can configure the 'resources' setting to control the amount of

CPU and/or memory the containers use. The framework can take significant time to process depending on the size of the data dump. A useful command to check on the status of containers is 'docker container stats'. This command shows the currently running containers and their resource utilization.

5.1. Current Functionality

To-date, the framework provides the capability to process six data types using a supplied list of keywords. The results are stored in text files which are grouped by keyword. The processing occurs concurrently. There is no known limitation for the depth of directories and subdirectories the framework can traverse or the number of files it can search through. The framework will also handle decompressing files and adding newly decompressed files into the processing workflow. The specific data types supported are as follows:

Searching

- ✓ Plaintext files
- ✓ Microsoft Excel spreadsheets
- ✓ Legacy Microsoft Excel spreadsheets
- ✓ Microsoft Word documents
- ✓ Portable Document Format (Pdf)

Decompression

- ✓ Gzip

5.2. Tested Platforms

The framework was developed and tested using Ubuntu 20.04.1. Though it technically works on Microsoft Windows using Docker Desktop, the performance is abysmal due to how Docker containers access the Windows file system. Docker Desktop for Windows is facilitated by using Windows Subsystem for Linux 2 (WSL2). The path to attach volumes from Windows to Docker is: Docker → WSL2 → Windows file

system. This connection path adds latency and renders the framework unusable. The latency for attached volumes is a known issue, and Microsoft generally recommends against "working across operating systems with your files" (Microsoft, 2020). Performance is likely better if attached volumes reside directly in WSL2 instead of the Windows filesystem (Docker, n.d.); however, this research did not cover this test. This framework was not tested with Docker Desktop on Mac.

6. Future Considerations

The current functionality of the framework establishes a foundation to build and improve on. There is support for six common data types, running in containers, and support for new data types can be built and plugged into the framework. There are, however, many usability and optimization improvements that could be implemented to make the framework more approachable and beneficial to the security community.

The framework would benefit from a web interface to set configurations, initiate searching, view progress, and view results. A web interface would make it much easier to start processing and provide continuous feedback rather than waiting for completion. There are also new search features to consider. For example, regular expressions would allow for standard data formats like email addresses or social security numbers to be identified.

Additionally, there are opportunities to optimize the search speed. Using a database to manage state has introduced race conditions wherein one processor attempts to work on a file that another processor has already started. There are checks built into the application to avoid this, but in some cases, the result is duplicate processing of the same file. The race conditions do not cause errors or duplicity in the results, but they do waste processing resources and increase the total processing time. Complimenting the database with a queue mechanism may elevate duplicate processing issues, but it requires further exploration.

7. Conclusion

Processing data dumps is a data problem at its core. Security professionals may not always have the time and/or resources to address this problem. The solution requires a flexible approach, which Docker containers can afford. The framework presented in this article provides core functionality for common data types found within data dumps and the foundation to build upon. Utilizing the framework can enable the identification of interesting data points to improve information security defenses.

References

Aghlara, A. (2020, January 27). *What is champion Challenger and how does it enable choosing the right decision?* Medium. https://medium.com/decision-automation/what-is-champion-challenger-and-how-does-it-enable-choosing-the-right-decision-f57b8b653149

Benevides, R. (2016, February 24). *10 things to avoid in docker containers*. Red Hat Developer. https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/

Berlioz, C. (2014, February 10). *What are Champion / Challenger experiments in Decision Management?* Sparkling Logic. https://www.sparklinglogic.com/what-are-champion-challenger-experiments-in-decision-management/

Docker. (n.d.). *Python | Docker Official Images*. Retrieved February 9, 2021, from https://hub.docker.com/_/python/)

Docker. (n.d.). *Docker hub*. Retrieved February 9, 2021, from https://hub.docker.com/search?image_filter=official&type=image

Docker. (n.d.). *What is a container?* https://www.docker.com/resources/what-container

Docker. (n.d.). *Overview of docker compose*. https://docs.docker.com/compose/

Frost & Sullivan. (2017). 2017 Global Information Security Workforce Study. Retrieved from https://www.iamcybersafe.org/wp-content/uploads/2017/06/EuropeGISWS-Report.pdf

Docker. (n.d.). *Docker desktop WSL 2 backend*. https://docs.docker.com/docker-for-windows/wsl/

Google. (n.d.). *What are containers and their benefits | Google cloud*. https://cloud.google.com/containers/

Hayes, A. (2011, December 4). *The importance of the foreign key constraint*. DBA Diaries. https://dbadiaries.com/foreign-key-constraints

Krebs, B. (2020, June 22). *'BlueLeaks' exposes files from hundreds of police departments*. Krebs on Security. https://krebsonsecurity.com/2020/06/blueleaks-exposes-files-from-hundreds-of-police-departments/

TJ Nicholls, Packetvitality@gmail.com

Hunt, T. (n.d.). *Pastes*. Have I been Pwned. https://haveibeenpwned.com/Pastes

Jagtap, M. (2019, April 3). *What is SQLite? – SphereGen*. SphereGen. https://www.spheregen.com/sqlite/

KPI. (n.d.). *What is a key performance indicator (KPI)?* https://kpi.org/KPI-Basics

Malwarebytes. (n.d.). Data breach - What is it and how to handle. https://www.malwarebytes.com/data-breach/

Maple Tech. (n.d.). *Sample size calculator*. Calculator.net: Free Online Calculators. https://www.calculator.net/sample-size-calculator.html

Nishanil. (2018, August 31). *What is docker?* Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined

Oracle. (n.d.). *What is a relational database?* https://www.oracle.com/database/what-is-a-relational-database/

OWASP. (2020, November 28). Credential stuffing. https://blog.owaspvit.com/2020/11/credential-stuffing.html

RenditionSec. (2017, August 13). The need for dump analysis in Cyber Threat Intelligence (CTI). Retrieved September 22, 2020, from https://blog.renditioninfosec.com/2017/08/the-need-for-dump-analysis-in-cyber-threat-intelligence-cti/

PyPi. (n.d.). *Python-magic*. Retrieved February 9, 2021, from https://pypi.org/project/python-magic/

SQLite. (n.d.). *SQLite frequently asked questions*. https://sqlite.org/faq.html

SQLite. (n.d.). *SQLite foreign key support*. https://sqlite.org/foreignkeys.html

SQLite. (n.d.). *Write-ahead logging*. https://sqlite.org/wal.html

Timby, B. (2018). *fulltext*. Retrieved February 9, 2021, from https://github.com/btimby/fulltext

Tutorialspoint. (n.d.). *Computer programming - Functions*. Retrieved February 9, 2021, from https://www.tutorialspoint.com/computer_programming/computer_programming_functions.htm

TJ Nicholls, Packetvitality@gmail.com

Verizon. (2020). *2020 Verizon Data Breach Investigations Report*. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf