File Analysis: Forensics in the Cloud

Vibha Iyengar, Arijeet Mitra, Sireesha Pilaka
Carnegie Mellon University

***Abstract*— Analyzing a large set of files is a common use case in forensic investigations. Manually doing this is a daunting task and there is no reason to not use readily available tools to make the problem more tractable. In this project, we explore leveraging cloud-computing resources in building a fast, easily deployable, scalable pipeline to automate the process. We also proceed to evaluate the pipeline against BinaryAlert, an open source incident response AWS pipeline in terms of processing speed, flexibility, modularity and minimal processing.**

***Keywords*— Digital Forensics, Cloud Forensics, File Analysis, AWS Pipeline, Serverless Architecture**

I. Introduction

Forensic investigators often need to go through a huge amount of files to identify potential evidence for a digital crime. Companies need to be able to perform file analysis in real-time on the data going in and out of the internal network infrastructure to avoid a potential malware infestation or leaking a confidential piece of information etc.

Manually reversing files is not feasible to do especially when the input data set is large or when time is a very sensitive factor like in the case of email attachments. There are numerous tools and techniques to do file analysis and to separate out the known good or bad files. And using a combination of such tools in an automated pipeline on the corpus of files would make a forensic investigator's job a much more tractable problem.

But setting up the infrastructure on physical servers to run the tools would take a lot of time and manual effort.The resources allocated such as servers, databases etc. are dedicated machines performing this task. So, this is not necessarily a cost-effective solution. It is extremely difficult to scale the resources as per the size of file corpus resulting in a very sub-optimal utilization. Updating versions, configurations, adding or removing tools from the pipeline are quite challenging since the process lacks modularity. Moreover if one of the tools comes down, it can break the entire process. In other words, the tools could potentially become single points of failure.

Traditionally, DevOps teams have had a very similar problem to deal with as well. Especially as the industry started migrating towards microservices architectures, infrastructure teams had more and more work at hand. Many organizations today leverage cloud services and abstract most of the infrastructure management and maintenance away to a cloud-service provider (CSP) so that they can focus more on developing their products and not on managing the platforms for development.

Serverless architectures take this one step further and run applications in stateless compute containers. These containers are fully managed by the CSP in terms of scaling, updates, optimal utilization etc. as well along with the rest of the infrastructure, whereas in other typical Infrastructure-as-a-Service(IaaS) offerings, the container layer is not handled by the CSP. [1][2]

The ephemeral compute power is offered on request and disappears after use. The functions are split into individual services and run which makes deploying changes to each module or component a significantly simpler task. Scaling the services is automatic and completely taken care of by the CSP. AWS(Amazon Web Services) *Lambda* is one of the most commonly used serverless offerings. [5]

Leveraging such cloud-computing capabilities seems to be a clear solution to our problem of building and managing highly scalable, cost and resource efficient, easily deployable and modifiable, modular forensics pipelines. The popularity of BinaryAlert, a real-time AWS pipeline for detecting malicious files, validates this intuition. [4]

Placing digital forensics tools in the cloud would mean investigators do not need to spend time installing numerous tools, manually feeding inputs and managing the outputs for each of the tools. The process would be entirely automated and based on the design, scalability and modularity would be taken care of as well.

In this project, we build a framework in AWS to help automate file analysis. We formally present the problem statement, explain our architecture and design decisions and then we proceed to evaluate this pipeline against another AWS pipeline, BinaryAlert.

II. Problem Statement

In this project, we are building a pipeline in AWS, leveraging the cloud capabilities to enable easier and faster file processing with capabilities to identify malicious files, to classify files into known good files, bad files and unknown files. We trigger alerts when bad files are detected. The unclassified files can be ignored or sent to further analysis. We aim to make the pipeline easily deployable and the components, their configurations, report mechanism customizable to the internal needs of any forensics case or incident response team of an organization. We chose to build our pipeline in AWS since it is a widely used platform with relevant offerings to our use case such as Lambda. [5] [6]

III. Related Work

BinaryAlert is an open-source AWS pipeline built by Airbnb to detect malicious files based on YARA rules. These YARA rules describe patterns or signatures exhibited by various malware families and when a match is found to one of the rules, an alert is triggered. [3][4]

BinaryAlert is built for real-time malware detection to help internal incident response teams of organizations avoid the spread of malware. While BinaryAlert focuses on primarily malicious files, we are building a more general purpose framework which is also suitable for detecting any pre-configured patterns or types of files, not just the malicious ones. This is can give many advantages such as helping with cleaning out known good files in the process, speeding up the process, triggering alerts for other interesting non-malicious files (Eg: internal confidential documents being emailed out to external entities) etc.

The feedback mechanism in our framework enables the pipeline to learn the category of processed files and thereby improve performance when it comes to identifying either similar files or the same ones again. For instance, if an organization wishes to label a large set of files from various hosts, the corpus would contain a big chunk of system and library files repeated from each machine which are known good files. We can save processing power when it comes to these files by adding hash comparison with a database and a feedback mechanism to evolve the database over time to improve performance and finetune needs and focus on a specific type of file.

IV. Tools and Techniques

Reverse engineering each file to classify or label the files is not an easy task. It consumes a great deal of time and resources. More often than not, someone in the broader security community would have already performed such an analysis of the file and shared any findings which make the analysis of the file redundant. There are large databases of file metadata such as OWASP's file hash repository[7], VirusTotal[8] etc.. The community as a whole has also observed several patterns or signatures in different categories of files and documented the same. Many file analysis tools leverage this information in categorizing files. Some of the well-known techniques these tools use are hash matching and YARA rules based pattern matching.

A. Hash matching

Hash matching involves generating a cryptographic hash for the file and matching the hash value against a set of known hashes to identify the file. The technique is based on the assumption of computational infeasibility of finding hash collisions.

We are using hash matching with a known set of hashes present in the database (MySQL) from OWASP's file hash repository[7] to recognize known good or bad files. As the first step in our pipeline, this filters out some files and thereby avoids further processing which reduces overall latency.

In the MD5 algorithm, hash collisions have been found which breaks the assumption of this technique that the hashes are unique per file.[9] Also, the database can not contain an exhaustive list of file hashes which leads to a decent number of files not being identified at all. The hash value of a file gets severely modified even if a single bit is flipped and most of the time this kind of a modification is used in malware to avoid detection. So, using just hashes to identify files is clearly not going to be sufficient. Moreover, the overall success of the approach hinges on having a mature database with entries corresponding to as many known files as possible.

B. YARA rules

YARA rules are rules which define patterns and conditions to identify and classify families of files based on certain behaviors. [10] Typically, most malware behaviors such as having specific strings, referring to known bad ip addresses or sink holes etc. are pretty well known. A great deal of such patterns are identified and are included in many YARA rules-based tools.

YARA rules are also quite simple to define and can be defined to track or identify different kinds of files as per the use case which grants a great deal of flexibility to the users.[12] These behavior patterns remain even if the file is severely modified or even rewritten where hashes don't match. For instance, some malware checks for the presence of a debugger which is not a common behavior in most benign programs. So, the presence of "DebugActiveProcess" kind of strings etc. could indicate that the file is malicious. [11]

The efficiency of the technique depends on the vastness of the set of rules being used. Since the rules might check for patterns within the file, this could take much longer than hash comparison.

C. Context-triggered piecewise hashes or Fuzzy hashes

Piecewise hashes involve creating hashes for chunks of data in a file instead of one single hash for the entire file. The Context-Triggered Piecewise Hashing (CTPH) algorithm uses a rolling hash to set the boundaries of traditional piecewise hashes. And, these fuzzy hashes computed are used in determining file matching instead of regular hashes.[13] This technique has been developed to mitigate the problem of a hash not matching in case of even minimal modifications.

Though the approach fixes one problem of the hash matching approach, the efficiency still remains dependant on the comprehensiveness of the entries in the database since for the approach to be able to have any meaningful results, the database must contain an entry pertaining to the file. Computing fuzzy hashes takes longer than most traditional algorithms such as MD5, SHA256 etc. [13]

D. Hybrid approach

We are incorporating tools based on each of these approaches to improve the performance and get the maximum utilization of the collective knowledge the security community as a whole has accumulated over time.

E. Feedback mechanism

In the database, we have two tables - one with the MD5 and SHA256 hashes and another with fuzzy hashes. We are using a tuple of (MD5, SHA256) hashes as the primary key in the table of hashes to reduce the probability of running into hash collisions. And each time a file is detected by YARA rules, the hashes are computed for the file and these values are included in the tables.

If another file which matches the hash value enters the pipeline, it can be marked as good or bad at the hash matching phase itself. This is intended to reduce the amount of processing needed for any similar or the same files showing up in the future. This also helps fine tune the database over time to suit specifically to the users' needs.

F. Modular design

We are keeping our pipeline very modular - as in each component or tool is run as an independent function in a Lambda so as to maintain the scalability. This also makes adding any other tools, updating or removing the existing tools in the pipeline significantly easy.

The user can add any additional capabilities at any stage that suit his use case. We are adding a tool called Ghiro for processing images[14] to showcase the modularity of the pipeline and how easy it is to add or remove a tool from the pipeline.

V. System Architecture

The system architecture is shown in Figure 1. It is a pipeline deployed in AWS infrastructure to automate forensics evaluation in the cloud.

Our pipeline makes use of S3 (Simple Storage Service) and Lambda integration provided by AWS where S3 monitors the bucket for events, and invokes

Lambda functions with the event as a parameter. In our case, the event is a file being added to the S3 bucket.

The forensic investigator needs to drop files into the S3 bucket, insure-raw, for persistent storage to start the process. Any incremental changes in the first bucket trigger[15] AWS Lambda functions to compute the hash of the file and this compares it with the values in the database for known malware and suspicious activities. It moves over the good/undetectable files over to next bucket, insure-bloom, for further processing by YARA analyzers.

YARA scan compares the signature of the files dropped in the second bucket against the compiled rules of known bad files. It moves over the rest of the good files over to next stage, insure-filtered bucket, for their fuzzy hash computation by ssdeep and comparing with known fuzzy hashes of bad files.

We are using a centralized MySQL database as a source of truth for known bad files hashes. There are separate tables for MD5, SHA1 hashes and fuzzy hashes in this architecture. Bad files detected at any stage in the pipeline is pushed into the database for future pruning at earlier stages.

The clean files from ssdeep evaluation are dropped into an S3 bucket, insure-final. Currently we are triggering a Lambda function, if any image files are added to this bucket. The Lambda moves the images to a separate workstation for image specific analysis. The user can add tools specific to any other type of files based on his needs by adding similar triggers on the insure-final S3 bucket.

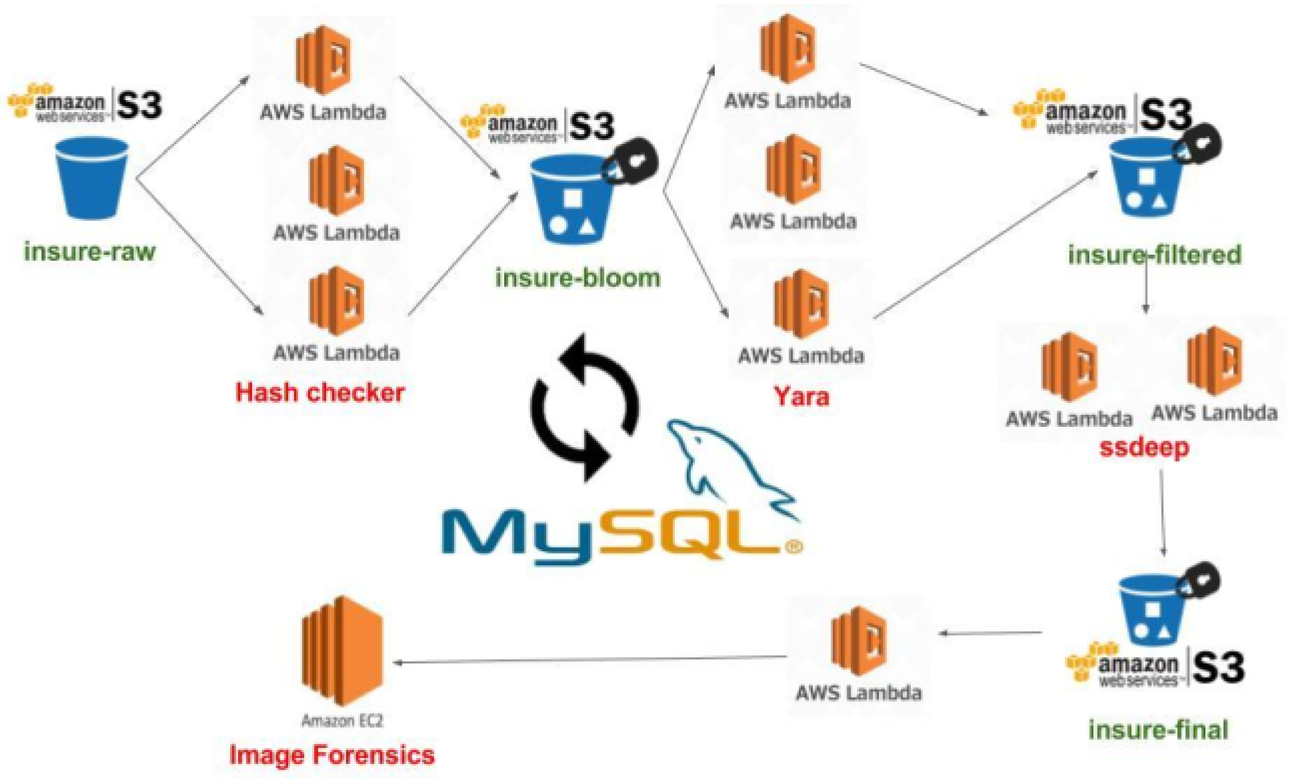

Fig. 1 Architecture: AWS pipeline for File Forensics

A. MD5Checker

This is a python library used to compute the md5 and sha1 hash of the file.[16] This is embedded in a Lambda which is the entry point of our pipeline. The result is compared against the known bad hashes in the database. It used the tuple of <md5, sha1> as the primary key to uniquely fetch details from the datastore.

B. YARA Scanner

The second component of the pipeline uses YARA rules to identify and classify malware using pattern matching.[10] The YARA rules allow us to do signature-based detection of malware against historic samples embedded with the YARA scanner.[17] This is uploaded to a secure persistent storage in S3 bucket to be used in parallel across Lambdas for processing. We compute the hash of the matched files and store it into the database.

C. ssdeep Checker

ssdeep[18] is a tool for computing context triggered piecewise hashes, commonly referred to as fuzzy hashes. It partitions the file through a rolling hash function, uses another hash function to compute the hash of each piece (piecewise hash) and then concatenate the result to produce the hash signature of the file.[13]

This tool used predefined levels of correlations to determine the similarity between the fuzzy hashes. This computed fuzzy hash is compared against the known bad fuzzy hashes in the datastore using this tool.

D. MySQL database

Historically known bad file hashes and fuzzy hashes are stored in a centralized MySQL database. The datasource is secured using AWS inbound rules to make it accessible only from specific hosts. At any stage in the pipeline, the hashes of the identified bad files are stored in the database in the appropriate tables. This allows an evolving database so that the pipeline becomes more mature with time while reducing the false positives at the earlier stages of the pipeline to save on compute power. MySQL comes with the capability of indexing to improve on query performance.[19]

E. Ghiro

Ghiro[14] is separate workstation exclusively for image forensics. We are adding this tool to demonstrate the modularity and flexibility of our pipeline. If the user intends to further process various types of files such as pdfs, MS Office files, images etc. separately, he can add any relevant tools at this stage in the pipeline.

The Ghiro workstation is triggered on any uploads on the last S3 bucket through Lambda functions. The tool performs image specific analysis tasks such as recognizing thumbnail inconsistencies, different compression rates across the image etc.

F. FileIntel

FileIntel[20] is a tool to collect information about a file from various intelligence sources. We have configured it to get the information on the file from VirusTotal scan and this information is collected when an alert gets triggered and is sent along with the alert.

VI. Evaluation Results

We set up the tool BinaryAlert[3][4], described in Section III and compared the performance of both the tools and the results are documented in Table 1.

TABLE I
Comparative Analysis

Criteria	**BinaryAlert**	**Proposed Pipeline**
Time required for analysis of a file	Usually completed within 1–2 minutes of file discovery. This is true for all types of files, and each and every time.	Time varies depending on the kind of file. Known malware files are detected earlier than the ones which are deceived malicious files. Max time around 2 minutes or lesser
Inclusion of YARA rules	Consists in-built YARA rules, and is compatible with open source resources like yararules[21] as well	Majorly uses YARA rules from open source projects like yararules[21], clamav-yara[22], and reyara[23].
Flexibility	Tightly bound structure, hence cannot eliminate any component based on use case[4]	Loosely coupled structure, enabling to configure the pipeline to suit the use case
Replayed files	Each time the file is placed for analysis, the entire process is repeated using YARA rules, consuming a long time	Consists a learning model. Files are matched against the locally stored hashes first. If found, further analysis is avoided. The result is

		alerted, consuming less time.
Serverless Design	Utilizes AWS Lambda functions for analysis instead of a traditional server.	Utilizes AWS Lambda functions for analysis instead of a traditional server.

VII. Future Work

In our current design, the MySQL database can act as a single point of failure. Replacing this with a database cluster would mitigate this problem.[24] Since we expect our database to expand with time due to our feedback mechanism, there is a need to determine reasonable limits on how much it should be allowed to grow and what data can be evicted from the database to make room for the new data. Another possible next step would be exploring additional tools such as PDFExaminer[25] to examine pdf files, OfficeMalScanner[26] for malicious MS Office documents etc. in the pipeline to process different kinds of data similar to the image forensics by Ghiro because the end goal is to classify as many files as possible.

VIII. Summary Of Features

Flexibility: Decoupled components in the pipeline allow users to configure i.e. add/remove components.

Learning Model: Storing the hash and metadata of bad files detected at any stage or by any components in the pipeline allows the pipeline to evolve. This makes sure that similar bad files are detected at earlier stages with time.

Lesser Time: Using the serverless technology to process chunk of files in parallel is similar to BinaryAlert. However, the fusion of evolving datastore and delegation of malware identifier based on file type improves the performance and provides an added advantage.

Lesser False Positives: Embedding components with workstations for different types of bad files allows better classification and analysis. We incorporate checking hashes, fuzzy hashes and image forensics over the classic YARA signature detection that BinaryAlert provides.

Automation: Build and deployment of lambdas through scripts which could be configured by users.

IX. Conclusion

Our proposed framework is fast, resource efficient, easily deployable, modifiable, scalable and can be configured and/or bootstrapped to the user's needs. It inherits the powerful capabilities of cloud-based architectures and has use cases in incident response and forensic investigations.

References

[1] Thoughtworks.com. (2017). *Serverless architecture*. [online] Available at: https://www.thoughtworks.com/radar/techniques/serverless-architecture [Accessed 8 Dec. 2017].

[2] Roberts, M. (2017). *Serverless Architectures*. [online] martinfowler.com. Available at: https://martinfowler.com/articles/serverless.html [Accessed 8 Dec. 2017].

[3] Airbnb.io. (2017). *BinaryAlert | Airbnb Engineering*. [online] Available at: http://airbnb.io/projects/binaryalert/ [Accessed 8 Dec. 2017].

[4] Medium. (2017). *BinaryAlert: Real-time Serverless Malware Detection*. [online] Available at: https://medium.com/airbnb-engineering/binaryalert-real-time-serverless-malware-detection-ca44370c1b90 [Accessed 8 Dec. 2017].

[5] Amazon Web Services, Inc. (2017). *AWS Lambda - Serverless Compute*. [online] Available at: https://aws.amazon.com/lambda/ [Accessed 8 Dec. 2017].

[6] D1.awsstatic.com. (2017). [online] Available at: https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf [Accessed 8 Dec. 2017].

[7] Owasp.org. (2017). *OWASP File Hash Repository - OWASP*. [online] Available at: https://www.owasp.org/index.php/OWASP_File_Hash_Repository [Accessed 8 Dec. 2017].

[8] Virustotal.com. (2017). *Documentation - VirusTotal*. [online] Available at: https://www.virustotal.com/en/documentation/ [Accessed 8 Dec. 2017].

[9] Kessler, G. C. (2016). The Impact of MD5 File Hash Collisions on Digital Forensic Imaging. *Journal of Digital Forensics, Security and Law*, *11*(4), 9.

[10] Mosli, R., Li, R., Yuan, B., & Pan, Y. (2016, May). Automated malware detection using artifacts in forensic memory images. In *Technologies for Homeland Security (HST), 2016 IEEE Symposium on* (pp. 1-6). IEEE.

[11] Ortega, A. (2017). *Your malware shall not fool us with those anti analysis tricks*. [online] www.alienvault.com. Available at: https://www.alienvault.com/blogs/labs-research/your-malware-shall-not-fool-us-with-those-anti-analysis-tricks [Accessed 8 Dec. 2017].

[12] Virustotal.github.io. (2017). *YARA - The pattern matching swiss knife for malware researchers*. [online] Available at: https://virustotal.github.io/yara/ [Accessed 8 Dec. 2017].

[13] Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, *3*, 91-97.

[14] Docs.getghiro.org. (2017). *Ghiro's documentation — Ghiro 0.2.1 documentation*. [online] Available at: http://docs.getghiro.org/en/ghiro-0.2.1/ [Accessed 8 Dec. 2017].

[15] Docs.aws.amazon.com. (2017). *Using AWS Lambda with Amazon S3 - AWS Lambda*. [online] Available at: http://docs.aws.amazon.com/lambda/latest/dg/with-s3.html [Accessed 8 Dec. 2017].

[16] Pypi.python.org. (2017). *md5checker 0.1.1 : Python Package Index*. [online] Available at: https://pypi.python.org/pypi/md5checker/0.1.1 [Accessed 8 Dec. 2017].

[17] Yara.readthedocs.io. (2017). *Using YARA from Python — yara 3.4.0 documentation*. [online] Available at: http://yara.readthedocs.io/en/v3.4.0/yarapython.html [Accessed 8 Dec. 2017].

[18] Ssdeep-project.github.io. (2017). *ssdeep - Fuzzy hashing program*. [online] Available at: https://ssdeep-project.github.io/ssdeep/index.html [Accessed 8 Dec. 2017].

[19] A2hosting.com. (2017). *Using indexes to improve MySQL query performance*. [online] Available at: https://www.a2hosting.com/kb/developer-corner/mysql/using-indexes-to-improve-mysql-query-performance [Accessed 8 Dec. 2017].

[20] GitHub. (2017). *keithjjones/fileintel*. [online] Available at: https://github.com/keithjjones/fileintel [Accessed 8 Dec. 2017].

[21] GitHub. (2017). *Yara-Rules/rules*. [online] Available at: https://github.com/Yara-Rules/rules [Accessed 8 Dec. 2017].

[22] GitHub. (2017). *sec51/clamav-yara*. [online] Available at: https://github.com/sec51/clamav-yara [Accessed 8 Dec. 2017].

[23] GitHub. (2017). *VectraThreatLab/reyara*. [online] Available at: https://github.com/VectraThreatLab/reyara [Accessed 8 Dec. 2017].

[24] Docs.aws.amazon.com. (2017). *Creating a DB Cluster and Connecting to a Database on an Amazon Aurora DB Instance - Amazon Relational Database Service*. [online] Available at: http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.Aurora.html [Accessed 8 Dec. 2017].

[25] Pdfexaminer.com. (2017). *PDFExaminer: pdf malware analysis*. [online] Available at: http://pdfexaminer.com [Accessed 8 Dec. 2017].

[26] Reconstructer.org. (2017). *www.reconstructer.org*. [online] Available at: http://www.reconstructer.org/code.html [Accessed 8 Dec. 2017].