BinaryPig: Scalable Static Binary Analysis Over Hadoop

Zachary Hanif, Telvis Calhoun, Jason Trost
Endgame Inc, Atlanta, Georgia
{zach, tcalhoun, jtrost}@endgame.com

I. Abstract

Malware collection and analysis are critical to the modern Internet security industry. This increasing tide of 'unique' malware samples and the difficulty in storing and analyzing these samples in a scalable fashion has been frequently decried in academic forums and industry conferences. Due to the increasing volume of samples that are being discovered in the wild, the need for an easily deployed, scalable architecture is becoming increasingly pronounced. Over the past 2.5 years Endgame received 20M samples of malware equating to roughly 9.5 TB of binary data. In this, we're not alone. McAfee reports that it currently receives roughly 100,000 malware samples per day and received roughly 10M samples in the last quarter of 2012 [1]. Its total corpus is estimated to be about 100M samples. VirusTotal receives between 300k and 600k unique files per day, and of those roughly one-third to half are positively identified as malware [2]. Effectively combating and triaging malware is now a Big Data problem. Most malware analysis scripts are unprepared for processing collections of malware once they surpass the 100GB scale, let alone terabytes. This paper describes the architecture of a system the authors built to overcome these challenges that uses Hadoop [13] and Apache Pig [14] to enable analysts to leverage pre-existing tools to analyze terabytes of malware, at scale. We demonstrate this system to analyze ~20M malware samples weighing in at 9.5 TB of binary data.

II. Introduction

Malware authors utilize multiple methods of obfuscating their executables to evade detection by automated anti-virus systems. Many malware authors utilize packing, compression, and encryption tools in an effort to defeat automated static analysis; authors continually pack and repack their samples and check them against automated scanning engines. These samples are shared between interested parties, thereby inflating the number of samples that require analysis. These samples are relatively low value: many of them are never released into the wild, and moreover, as their core functionality has not been altered, analysis of these samples represents a needless redundancy. For samples that are in the wild, autonomously spreading malware often have algorithms which alter key elements of their executable payloads before transmission and execution on a newly compromised system in an effort to evade detection from network security sensors and other simpler protections. Collection of autonomously spreading samples by honeypot systems such as Amun [2] and Dionaea [3] therefore collect huge numbers of samples that appear unique when compared using cryptographic file checksums, yet represent functionally identical software. Further complicating the issue are the samples that are collected by both systems that represent false positives: samples that have mistakenly triggered a heuristic, or benign samples that were submitted to an online scanning engine.

BinaryPig hopes to provide a solution to the scalability problem represented by this influx of malware. It promotes rapid iteration and exploration through processing non-trivial amounts of malware at scale using easily developed scripts. It was also built to take advantage of pre-existing malware analysis scripts.

III. Prior Work

While there is no lack of published descriptions for malware analytic models and feature sets, to our knowledge, there are no salient papers that discuss the architecture behind the feature and data extraction from a static perspective [7], [8], [9]. As distributed data processing has engineering concerns that are generally unseen in centralized processing infrastructure, and the majority of papers that we explored while researching during the construction of this system focus on the results of derived analytics, it is assumed that the systems which coordinated the extraction, results and sample storage, and other distributed processing concerns were either considered to be too simplistic, too specialized, or too ungainly for publication. As these papers have been published, however, the authors believe that while systems similar in function to BinaryPig exist, our offering represents something novel in the form of an extensible, free, open-source framework [10].

The most prominent papers surrounding data extraction infrastructure are represented in a tangential area of research: malware dynamic analysis. There have been a number of excellent discussions surrounding various methods of dynamic analysis at relatively large scales [8], [9] as well as work that lends itself to simply adding new analytical methods to an existing framework [18], none of this work takes advantage of architecture that is likely to exist within an organization that deals with large sums of data - essentially, again the extraction and storage infrastructure that drives many papers that have been published recently appears to be unique to each of the research institutions.

IV. Design

We needed a system to address scalability, reliability, and maintainability concerns of large-scale malware processing. We determined that Apache Hadoop met many of these requirements, but it had some shortcomings that we had to address.

Developing native MapReduce code can be time consuming even for highly experienced users. An early version of this framework was developed using native MapReduce code, and while functional, was difficult to use and difficult to extend. As a result, we opted to use the Apache Pig DSL to provide a more accessible user and developer experience.

During the development of BinaryPig, Hadoop's issues with storing and processing large numbers of small files was made apparent: attempting to store and iterate on malware samples directly resulted in memory issues on the NameNode and unacceptable performance decreases due task startup overhead dominating job execution times. To resolve these issues, we created ingest tool for compressing, packaging, and uploading malware binaries into Hadoop. The ingest tool combines thousands of small binary files into one or more large Hadoop SequenceFiles (each file typically ranging from 10GB to 100 GB in size), an internal data storage format that enables Hadoop to take advantage of sequential disk reads speeds as well as split these files for parallel processing [15]. The SequenceFiles created by BinaryPig are a collection of key/value pairs where the key represents the ID of the binary file (typically MD5 checksum) and the value is the contents of the file represented as raw bytes.

To ensure that the malware data was properly distributed across computing nodes in the cluster and resilient to failures, the Hadoop Distributed File System (HDFS) was utilized [19]; HDFS provides

many nice features for managing files and data including automatic data replication, failover, and optimizations for batch processing.

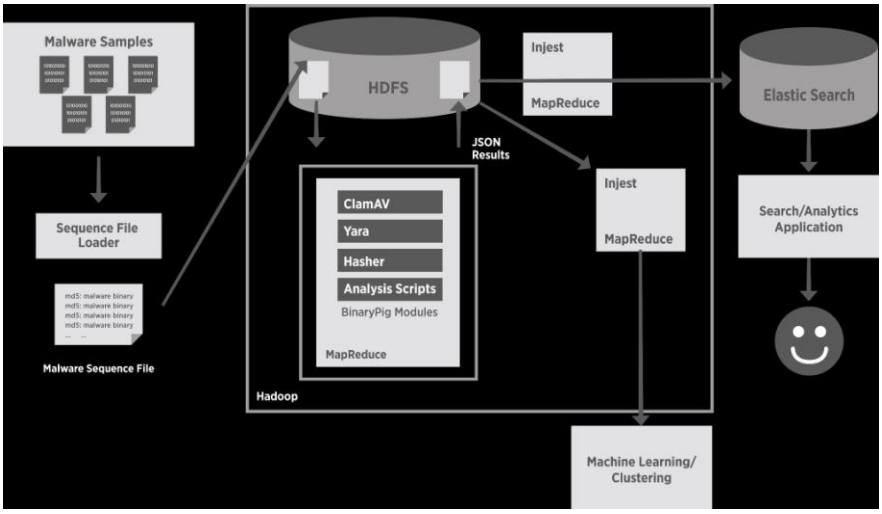
Within Pig, analysis scripts and their supporting libraries are uploaded to Hadoop's DistributedCache [16] before job execution and destroyed from the processing nodes afterwards. This functionality allows users to quickly publish analytical scripts and libraries to the processing nodes without the general concerns of ensuring deployment operated as expected or the time associated with a manual deployment.

We developed a series of Pig loader functions that read SequenceFiles, copy binaries to the local filesystem, and enable scripts to process them. These load functions were designed to allow these analysis scripts to read the files dropped onto the local filesystem. This allows BinaryPig to leverage pre-existing malware analysis scripts and tools. In essence, we are using HDFS to distribute our malware binaries across our cluster and using MapReduce to push the processing to the node where the data is stored. This prevents needless data copying as seen in many legacy cluster computing environments.

The framework expects the analysis scripts to write their output to stdout and to write any error messages to stderr (as most legacy malware analysis scripts already do). Ideally the data written is structured (JSON, XML, etc), but the system gladly handles unstructured data too.

Data emitted from the system is stored into HDFS for additional batch processing as needed. We also typically load this data into ElasticSearch indices [17]. This distributed index is used for manual exploration by individual users, as well as automated extraction by analytical processes. Through use of the Wonderdog library [12], we have been able to load data from the MapReduce jobs into ElasticSearch in a real-time fashion while continuing to serve queries.

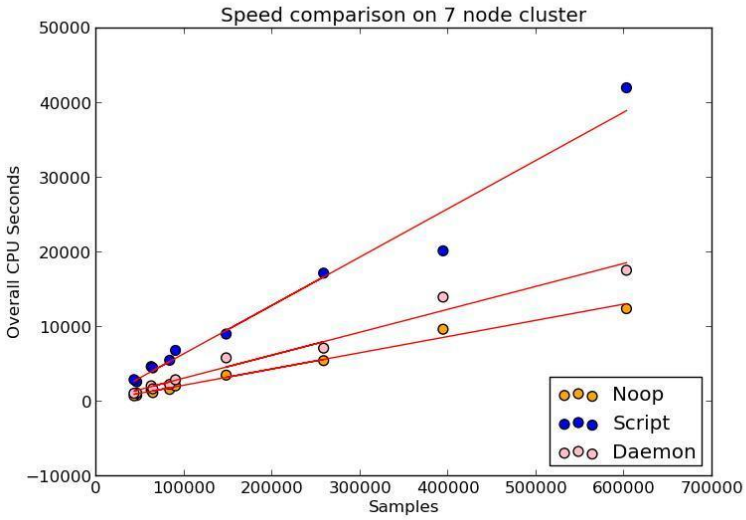
Fig 1 below provides a high-level diagram of the general architecture of the BinaryPig system.



We implemented optimizations to the backend processing system including using the /dev/shm/ virtual device for writing malware binaries. This device functions as a filesystem and allows for reading/writing files into memory managed by the OS. This allowed us to avoid a great deal of system and Disk I/O when writing files and creating processes that did not lend themselves to accepting

serialized files through stdin. Another optimization we implemented for some processing was creating long running analytics daemons in lieu of scripts. This drastically increased the performance for some important processing tasks since scripts are great for rapid iteration, but we found that the script interpreter startup times were dominating task execution times. The long running daemons listen on a socket, and read local file paths through this connection. They then analyze these respective files and return the results through the socket connection.

Fig 1 Performance comparison between scripts and daemons.



V. Preliminary Results

The initial testing and development of BinaryPig was performed on a 16-node Hadoop cluster. We loaded 20 million malware samples (~9.5 TB of data) onto this cluster. Both development time and execution time of new analytics are demonstrably faster, and the ability to execute them across historical sets, something that was not feasible before, is now possible. The ability to develop new analytics and execute them against historical sample sets for comparison and learning is exceptionally valuable. Full execution over the 20 million historical set was clocked at 5 hours when calculating the MD5, SHA1, SHA256, and SHA512 hashes for each sample. Operation speed is primarily based on the number of computational cores that can be dedicated to the process. Additionally, the ability to store, operate and analyze samples on a multi-use cluster is a marked improvement over requiring a dedicated cluster and storage environment.

VI. Ongoing Work

BinaryPig continues to be developed. Currently there is an effort to migrate currently existing analytics over to the new architecture so as to complete a malware census for use in developing predictive models. These results are continually developed and are intended for publication after validation. Additional input and output functionality between the Pig and analytics layers is being explored, as there is a need to be able to retrieve emitted binary data, without an intermediate encoding and serialization step, for icon, image, and other resource extraction. Additionally, we feel that the error

emission over the stderr interface is not optimal for production job management and subtle error catching, and are looking to expand the published examples with scripts that demonstrate such practices.

VII. References

[1] Higgins. “McAfee: Close To 100K New Malware Samples Per Day In Q2.” http://www.darkreading.com/attacks-breaches/mcafee-close-to-100k-new-malware-samples/240006702
[2] “Virus Total File Statistics.” https://www.virustotal.com/en/statistics/ as of 4/9/2013
[3] Amun, http://amunhoney.sourceforge.net/
[4] Dionaea, http://dionaea.carnivore.it/
[5] Chau, Duen Horng, et al. "Polonium: Tera-scale graph mining for malware detection." *Proceedings of the second workshop on Large-scale Data Mining: Theory and Applications (LDMTA 2010), Washington, DC*. Vol. 25. 2010.
[6] Perdisci, Roberto, Andrea Lanzi, and Wenke Lee. "Classification of packed executables for accurate computer virus detection." *Pattern Recognition Letters* 29.14 (2008): 1941-1946.
[7] Neugschwandtner, Matthias, et al. "Forecast: skimming off the malware cream." *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011.
[8] Rieck, Konrad, et al. "Automatic analysis of malware behavior using machine learning." *Journal of Computer Security* 19.4 (2011): 639-668.
[9] MTRACE, http://media.blackhat.com/bh-eu-12/Royal/bh-eu-12-Royal-Entrapment-WP.pdf
[10] Firdausi, Ivan, et al. "Analysis of machine learning techniques used in behavior-based malware detection." *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*. IEEE, 2010.
[11] Jang, Jiyong, David Brumley, and Shobha Venkataraman. "Bitshred: feature hashing malware for scalable triage and semantic analysis." *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011.
[12] Wonderdog. InfoChimps. https://github.com/infochimps-labs/wonderdog
[13] Apache Hadoop. Apache Foundation. http://hadoop.apache.org/
[14] Apache Pig. Apache Foundation. http://pig.apache.org/
[15] Sequence Files Format. Hadoop Wiki. http://wiki.apache.org/hadoop/SequenceFile
[16] DistributedCache. Apache Foundation. http://hadoop.apache.org/docs/stable/mapred_tutorial.html#DistributedCache
[17] ElasticSearch. http://www.elasticsearch.org/
[18] MASTIFF. http://sourceforge.net/projects/mastiff/
[19] Borthakur. HDFS Architecture Guide. http://hadoop.apache.org/docs/stable/hdfs_design.pdf