Machine Learning Toolkit for System Log File Reduction and Detection of Malicious Behavior

Ralph P. Ritchey
Department of Electrical and Computer Engineering
Villanova University
Villanova, PA, USA
rritche1@villanova.edu

Dr. Richard Perry
Department of Electrical and Computer Engineering
Villanova University
Villanova, PA, USA
richard.perry@villanova.edu

***Abstract*—The increasing use of encryption blinds traditional network-based intrusion detection systems (IDS) from performing deep packet inspection. An alternative data source for detecting malicious activity is necessary. Log files found on servers and desktop systems provide an alternative data source containing information about activity occurring on the device and over the network. The log files can be sizeable, making the transport, storage, and analysis difficult. Malicious behavior may appear as normal events in logs, not triggering an error or another obvious indicator, making automated detection challenging. The research described here utilizes a Python-based toolkit approach with unsupervised machine learning to reduce log file sizes and detect malicious behavior.**

Keywords—singular value decomposition, SVD, k-means, cybersecurity, log files, reduction, malicious behavior

I. Introduction

To reliably detect malicious activity, signature-based IDSs perform deep packet inspection on unencrypted network traffic. In a 2018 study conducted by Fortinet, Inc., "…encrypted traffic now represents over 72% of all network traffic…" [1], an increase of 17% from the previous year. The ability for signature-based IDSs to perform effectively decreases with each passing year as encryption use increases. The increasing use of encrypted network traffic impacting IDSs highlights the need for alternative data sources to perform routine, automated cybersecurity monitoring and analysis. Without the ability to monitor activity, cybersecurity analysts cannot perform their work, ensuring an organization is secure or detecting compromised systems in their environment.

Server and desktop log files are rich sources of information "…consist[ing] of the voluminous intermixing of messages from many software components..."[2] Transporting, analyzing, and storing the log files, even for a single system, can be challenging due to the large number of log lines some may contain. Logged cybersecurity events can appear as expected, non-error/fault entries, making detection challenging, an aspect not covered by other research papers. We propose a plugin-based Python toolkit[3] to effectively identify log lines indicating malicious behavior and reduce the log size down to more manageable sizes. The toolkit successfully chained truncated SVD with K-means to identify malicious log entries and significantly reduce log files. Plugins [4] for each workflow step allowed multiple variations to exist while improvements were made and tested.

II. Data Sets

Two different data sets were used, a publicly available set of log files and a private set. The public data set includes labels and allows other researchers to recreate the results presented here. While not releasable to the public, the private data confirms the research approach transfers to real-world data.

A. Public Data Set - Synthetic

Experiments used the Apache access log files from the AIT Log Data Set [5] provided by the Austrian Institute of Technology. The data set included labels, aiding the development, testing, and tuning of the approach in applying machine learning to log files. *Table I* contains the basic statistics of the number of log lines in each Apache access log file and the total number of log lines labeled as malicious. A successful attack against a web server results in the corresponding log line having a status code of 200, or "OK," indicating the server received, processed, and returned results without encountering a problem. A successful application of machine learning must correctly detect and classify malicious log entries with a non-error 200-status code.

TABLE I. AIT Data Set basic statistics

Server	Log Lines	Malicious Lines	Malicious/200 Status
mail.cup.com	148,534	6,789	475
mail.insect.com	169,340	6,973	665
mail.onion.com	81,963	6,429	129
mail.spiral.com	100,445	7,370	1,047

B. Private Data Set – Real-world

The real-world, private data set contained unlabeled, Apache access log files. The same data transformations and machine learning algorithms were used with a slight adjustment to the file processing plugin due to a difference in the logging format. This data set verified the approach worked when applied to real-world log files and not just a synthetic generated set. Classification accuracy was based on labels developed for this data set, however this data set did not contain 200-status log lines indicating successful, malicious behavior.

TABLE II. PRIVATE DATA SET BASIC STATISTICS

Server	Log Lines	Malicious Lines
Alpha	180,782	18,990
Beta	72,488	15,671
Gamma	68,442	18,476
Delta	438,208	57,505

Fig. 1. AIT cups server results

C. *Feature Extraction*

Initial processing of each log line extracted the basic features, including client IP, the number of bytes returned from the server, referrer, client user-agent, request type, parameters, HTTP protocol version, URL, and HTTP status code. Conversion of the status code translated it into a decimal value depending on the range the code belonged. For example, status codes from 200 to 299 inclusive translated to 0.2.

The request feature extraction had two variations. The first performed a simple split, breaking each request into the requested URL and parameter portion creating two features. The second variation extended the first by splitting the URL where the forward-slash occurred, generating a variable number of features. A maximum number of eight splits, starting from the left side of the URL and proceeding right, were performed.

An additional experiment built on the URL and parameter splitting by splitting the user-agent. Splitting the user-agent occurred where any semicolons appeared, after replacing any parenthesis with semicolons. Like spitting the URLs, this generated a variable number of features per log line up to a maximum of ten.

III. MACHINE LEARNING

Before applying the first machine learning algorithm in the chain, a new, machine learning ready matrix is created from the raw data matrix by hashing and scaling non-numerical values. Numerical values migrate into the new matrix without additional changes.

The first machine learning plugin used in the workflow, Truncated SVD, reduces the matrix's dimensionality to two dimensions. K-means, the second machine learning plugin, performs clustering, identifies the two centroids and performs classification using the reduced matrix.

The resulting output is a zero or a one, where one indicates the corresponding log line is malicious and retained for further analysis. Lines corresponding to a zero should be removed, resulting in a log file size reduction.

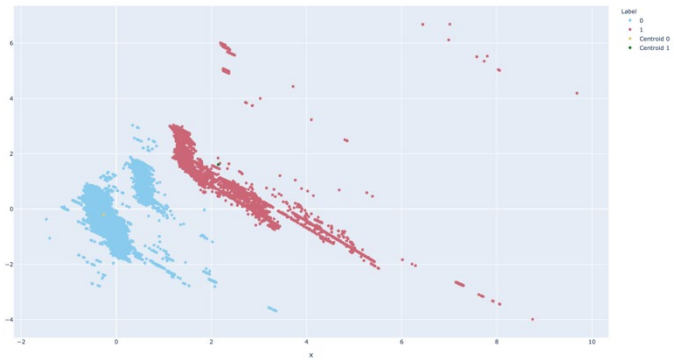

Fig. 2. Alpha Server Results

IV. RESULTS

Experiments used several variants of the file processing plugin for extracting features from the log files. Fig. 1 depicts a plot containing two K-means centroids centered within the two blue (non-malicious) clusters. Red plot points indicate labeled malicious data. The experiment splitting the URL and user-agent correctly identified 98.56% of the labeled malicious log entries, including 80.63% of malicious log entries with 200 status codes. Including false positives incorrectly identified as potentially malicious, log file reduction is 41.44%.

Fig. 2 depicts a similar experiment using real-world data from the alpha server. The file processor plugin splitting the URL into features results in 100% correct classification of the labeled malicious log lines, and an 88.95% reduction in log file size with false positives included. Without successful attacks included in the real-world log data, it is impossible to determine how well the approach correctly identifies malicious log entries with a non-error 200-status code.

Additional experiments monitored resource utilization to determine the feasibility of the approach for implementation as a real-world application. The real-world server named delta had the most extensive log file with 438,208 lines. The URL splitting approach used a maximum of 771MB of memory and approximately 50 seconds to process. These results indicate the method can be transferred from the laboratory environment to real-world application without undue resource requirements.

[1] Fortinet, Inc., "As the holiday season draws near, mobile malware attacks are prevalent," 14 November 2018. [Online]. Available: https://www.fortinet.com/blog/industry-trends/as-the-holiday-season-draws-near--mobile-malware-attacks-are-pre. [Accessed 26 June 2020].

[2] W.Xu, L. Huang, A.Fox, D.Patterson and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating System Principles*, Big Sky, MT 2009.

[3] R. P. Ritchey and R. Perrry, "MLTK-Log-Reduction-Detection", 17 February 2021. [Online]. Available: https://github.com/pritchey/MLTK-Log-Reduction-Detection. [Accessed 17 February 2021]

[4] R. P. Ritchey and T. W. Parker, "Simple plugin methodology in python," U.S. Army Research Laboratory, Adelphi, MD 2014.

[5] L. Max, S. Florian, W.Markus, H. Wolfgang and R. Andreas, "AIT Log Data Set v1.0 | Zenodo," 21 March 2020. [Online]. Available: https://zenodo.org/record/3723083#.Xv4HVy2z3UK. [Accessed 2 July 2020]