Semantic Interpretation of Structured Log Files

Piyush Nimbalkar
University of Maryland, Baltimore County
Baltimore, MD, USA
piyush4@umbc.edu

Varish Mulwad
GE Global Research
Niskayuna, NY, USA
varish.mulwad@ge.com

Nikhil Puranik, Anupam Joshi, Tim Finin
University of Maryland, Baltimore County
Baltimore, MD, USA
{puranik1, joshi, finin}@umbc.edu

***Abstract*—Data from computer log files record traces of events involving user activity, applications, system software and network traffic. Logs are usually intended for diagnostic and debugging purposes, but their data can be extremely useful in system audits and forensic investigations. Logs created by intrusion detection systems, web servers, anti-virus and anti-malware systems, firewalls and network devices have information that can reconstruct the activities of malware or a malicious agent, help plan for remediation and prevent attacks by revealing probes or intrusions before damage has been done. While existing tools like Splunk can help analyze logs with known schemas, understanding log whose format is unfamiliar or associated with new device or custom application can be challenging. We describe a framework for analyzing logs and automatically generating a semantic description of their schema and content in RDF. The framework begins by normalizing the log into columns and rows using regular expression-based and dictionary-based classifiers. Leveraging our existing work on inferring the semantics of tables, we associate semantic types with columns and, when possible, map them to concepts in general knowledge-bases (e.g. DBpedia) and domain specific ones (e.g., Unified Cybersecurity Ontology). We link cell values to known type instances (e.g., an IP address) and suggest relationships between columns. Converting large and verbose log files into such semantic representations reveals their meaning and supports search, integration and reasoning over the data.**

***Keywords*-log files; cybersecurity; knowledge graph; linked-data**

I. Introduction

Log files are composed of entries recording events associated with operating systems, applications running on devices, and events associated with networks to which devices are connected. All software and hardware devices running on systems generate large volume of log data. Nearly every event on a system triggers an entry of some data into a log file. While they were originally used to record information for debugging and diagnostic purposes, log files have evolved into recording events and information which is useful for audit trials and forensics in the event of malicious activities or system attacks. Recognizing the importance of logs, the National Institute of Standards and Technology, USA issued best practices and recommendations for computer security log management [1]. Similarly, enterprises have started devoting significant resources to build tools to manage and analyze large volumes of log data [2]. Current generation log file analyzers work well with logs from known devices. They are able to parse and interpret log files which have known formats and come from known sources, but fail to do so if the log file is of an unknown format/unknown source. It is often crucial to parse and interpret log files with unknown formats in the context of securing systems. In this paper, we present a framework that can infer the schema of log files from unknown sources and formats and generate a machine interpretable representation of log file data. This machine interpretable log file data can be integrated with other sources to reason over and detect and prevent cyber attacks.

Log files provide vital information about systems and the activities occurring on them. For instance, database logs can help trace how data was added and modified, while web server logs reveal patterns of how resources are accessed from the Web. On one hand, operating system logs help us figure out what is happening at the system level, while on the other hand, firewall logs help record malicious activities at the network level. Analyzing log files can thus provide valuable insights into system (mis)configuration as well as potential vulnerabilities. It is possible to use this information pro-actively to secure a system against potential malicious activities.

Large organization networks typically include a variety of computer security software and hardware devices spread over a number of hosts on the network each potentially generating multiple log files. Not only are a large number of log files generated, but the amount of data generated in each log is also massive. Traditional log file analyzers rely on knowing the format of the file ahead of time. Existing tools either focus on a specific type of log file (e.g. web server logs[1]) or several but a fixed set of log types[2]. Log files typically lack a common format with each log encoding data in its unique and often proprietary style. It is even more difficult to keep track of different log file formats in large heterogeneous networks in which software and devices are dynamic in nature; often new software and hardware are

[1]http://www.weblogexpert.com/
[2]http://www.splunk.com/

added to the network, each generating a log file with a unique schema. Existing approaches designed to deal with a fixed number of log files thus fail to scale in such scenarios.

Recent cyberattacks have been fairly sophisticated affecting different parts and applications as well as several systems simultaneously and are carried out over a period of several days. Detecting and preventing such low and slow vector attacks would require to generate a holistic view rather than analyzing each log individually. I One can obtain such holistic view by integrating information extracted from logs with information about security vulnerabilities from non traditional sources such as security blogs and bulletins. Threat prevention and detection in dynamically adapting scenarios thus requires integrating data from multiple traditional and non–traditional sources. Existing analyzers fail to generate data in a form that is suitable for integration and reasoning with other data sources.

```
10.4.8.16 aragorn [09/Dec/2015:19:22:26 -0500] "GET /favicon.ico" 404 498
10.4.8.16 aragorn [09/Dec/2015:19:22:30 -0500] "GET /web/people" 200 711
10.4.8.16 aragorn [09/Dec/2015:19:22:31 -0500] "GET /web/news" 200 814
10.4.8.16 aragorn [09/Dec/2015:19:22:31 -0500] "GET /icons/text.gif" 200 517
10.4.8.16 aragorn [09/Dec/2015:19:22:33 -0500] "GET /web/events" 200 745
```

Figure 1. Sample apache access log file with selected columns

We present an approach that infers the structure and the schema of the log file and uses the schema to generate a semantic representation of the log file content. Consider the example web server log in Figure 1. Each line in the log file records the same type information for a fixed set of fields. In our example, each line is recording IP address of the requesting host, user id of the requestor, time of request, requested resource etc. Our framework exploits this inherent structure in a log file to split it into distinct columns, with each column consisting of similar type of values. For example, the web server log can be split into columns such as Requesting IP Address, Requested Resource and so on. Based on the values in each column, our framework further infers and maps every column to a semantic class and every pair of columns to a semantic relation from a given ontology. Our framework further uses this mapping to generate a set of RDF triples from the log file content. These RDF triples can be integrated with other data sources to reason and detect potential security vulnerabilities [3].

The rest of the paper is divided as follows: we discuss related work in section II, go into the details of our approach in section III and present our evaluation in section IV.

II. Background and Related Work

We first provide background related to the Semantic Web technologies our work relies on and then discuss previous work related to our research.

We use Semantic Web standards such as RDF, OWL and Linked Data for capturing and representing knowledge in a machine interpretable format. Resource Description Framework (RDF) [4] is W3C "model for data interchange on the Web"[3]. RDF uses URIs to represent real world objects and properties or links to represent relationships between these objects. This representation is often called as a triple and consists of a subject, predicate (i.e. property representing the relationship) and an object. The triple representation of the statement "Barack Obama is the president of USA" would be <BarackObama> <isPresident> <USA>. The Web Ontology Language (OWL) [5] provides a vocabulary for defining real world concepts and relationships between them in the form of an ontology. The concepts in an OWL ontology are referred to as *classes* and relationships as *properties*. For example, concepts such as *President* and properties such as *isPresident* would be defined in an ontology. Linked Data [6] describes the best practices for publishing structured semantic data using URIs and RDF on the web to allow better interlinking, reasoning and querying of data. As of 2014, the Linked Data cloud comprises of 570 different knowledge bases belonging to a variety of domains such as Life Sciences, Social Networking, Government and User generated content.

Our work is related to two different threads of research – the first focuses on the use of text analytics to extract information related to cyber threats from unstructured texts and the second on automatically analyzing log files to detect and prevent cyberattacks. Joshi *et al.* [7] describe an automatic framework that generates and publishes a RDF linked data representation of cybersecurity concepts and vulnerability descriptions extracted from several structured vulnerability databases and unstructured text. This cybersecurity linked data collection was intended for vulnerability identification and to support mitigation efforts. The prime idea was to leverage the Linked Data representation of vulnerability databases to detect and prevent potential zero day attacks. Mulwad *et al.* [8] describe a framework to detect and extract information about attacks and vulnerabilities from Web text. They used Wikitology, a knowledge base derived from Wikipedia, to extract concepts related to vulnerabilities. Largely, work in this area has focused on generating semantic representations from structured databases or unstructured text from general sources such as blogs and security bulletins.

In the area of "log analysis", Nascimento *et al.* [9] use ontologies to analyze security logs. They create an ontology model from the ModSecurity logs and demonstrated that it was easier to find co-relations of events in log files when they were modelled using their ontology. Splunk [10] is a enterprise log analysis and management tool developed to analyze a large corpus of logs providing features such as searching, management, storage and visualization. It analyzes structured and unstructured log files and identifies

[3]https://www.w3.org/RDF/

fields in a given log file. It works well with log files whose structure it already knows. It is able to split a log file into fields, but fails to generate headers for log files from unknown sources.

III. Approach

Our approach builds on the concepts introduced in the *TABEL* framework [11], [12], [13] for inferring the semantics of information encoded in tables and representing it as RDF Linked Data. Figure 2 presents an overview of our system architecture. The *Tabulate* module parses and detects the structure of the log file. This module generates a table like structure by splitting the log into columns and rows. The *Decode* module identifies the type of values in each column and generates a ranked list of possible semantic classes from a given ontology. The *Relationship Generator* module uses this ranked list of classes, along with the ontology to additionally identify set of relations between the columns in the log file. Finally, the *Generate RDF* module finally generates a RDF Linked Data representation of the log file content using the inferred schema (classes and relations). We describe the four major modules in the rest of the section.

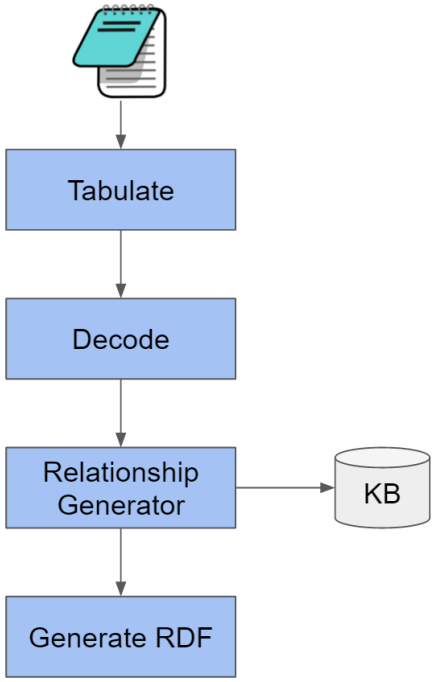

Figure 2. System Architecture

A. Tabulate

The Tabulate module parses and generate a table like structure for every log file by splitting it into columns and rows using set of regular expression patterns. The splitting algorithm's approach can be summarized into three steps:

- Split the log file into columns based on set of delimiters
- Detect consistent number of columns throughout the log file
- Detect and separate sub–columns within a column

In the first step, the algorithm splits every line in the log files using generic delimiters such as space, square braces, round braces, single and double quotes. Braces and quotes are used to split only when they appear in pairs. Every line (or row) in the file is thus separated into an inconsistent and varied number of columns.

In the second step, the algorithm normalizes the table structure by detecting consistent number of columns in the file. Although most of the lines in a log file have fixed number of columns, it is not necessary that delimiters will split the file into a table with same number of columns. Some log entries can be outliers due to the inconsistency in the way the logs are generated. In most of cases, there is a textual description column at the end of the log entry which does not have a fixed number of words. If the description is not enclosed in a brace or quote, it gets divided into several different columns. Taking these cases into consideration, we dynamically decide the number of columns that are to be extracted. The number of columns retained is equal to the maximum number of columns for at least 70% of the rows.

In the third step, the algorithm loops through the columns that are already separated in the previous steps and checks for sub–columns using the same set of delimiters. A column is split into multiple sub–columns only if all elements in the column are separable and if the Decode module fails to identify the type of values in that column.

B. Decode

The Detecting Commonly encoded Data Elements (Decode) module identifies the type of values in every column in the file and maps it to a semantic class from the Unified Cybersecurity Ontology (UCO) [14], [15]. The Decode module is based on Puranik's [16] "specialist" approach framework for identifying type of values in a given column. The original framework defines a specialist as "an entity that has expertise in the domain for which it is a specialist" and included three different types – regular expression based, dictionary based and machine learning classifier based specialists for identifying values appearing in general web tables such as Social Security Numbers (SSN), telephone numbers, airport codes, addresses and so on. Each specialist in this framework independently assesses a column and assigns a score of the likelihood that the column belongs to it's domain. For example, the SSN specialist assigns the likelihood that a given column consists of SSNs whereas the telephone specialist assigns the likelihood of the same values being telephone numbers. The framework then integrates all the specialist scores to generate a rank listed of possible value types. We extend the original framework to classify specific fields that are usually found in log files viz., timestamps, IP addresses,

URLs, etc. We add the following regular expression based and dictionary based specialists to the framework:

1) **Timestamp Specialist**
   The Timestamp specialist is a regular expression based specialist, which handles time stamps of different formats used in various systems for logging.
2) **IP Address Specialist**
   The IP Address specialist is a regular expression based specialist that checks for valid IP address formats and also makes sure that they are in a valid range.
3) **Port Number Specialist**
   The Port number specialist is a regular expression based specialist that checks for valid port numbers i.e. ones in the range of 0 to 65535.
4) **URL Specialist**
   The URL specialist is a regular expression based specialist, which looks for various URL formats viz., *HTTP*, *HTTPS*, *FTP*, *FTPS*. It also detects URL having basic authentication.
5) **Filepath Specialist**
   The Filepath specialist is also a regular expression based specialist. It looks for filepaths irrespective of the underlying system.
6) **Email Specialist**
   The Email specialist is a regular expression based specialist, that looks for valid email formats in the given column. It used the standard internet format to detect a valid email.
7) **HTTP Status Code Specialist**
   The HTTP Status Code specialist is dictionary based specialist. It looks for the HTTP status code values provided by the W3C standards. These HTTP status codes are integer values.
8) **HTTP Method Specialist**
   The HTTP Method specialist is a dictionary based specialist. It looks for the known HTTP verbs / methods like *GET*, *PUT*, *POST*, *DELETE*, etc.
9) **HTTP Protocol Version Specialist**
   The HTTP Protocol Version specialist is a dictionary based specialist, which looks for the known HTTP protocol versions. The HTTP protocol versions are in a string format.
10) **Log Level Specialist**
    The Log level specialist is a dictionary based specialist, which looks for the generally recommended log levels keywords. The log levels are in string format and contains log levels like *Info*, *Debug*, *Error*, *Warn*, etc.

As in the original specialist framework, every column in the log file is processed by each of the above specialists. Each specialist assigns a score to every column based on the number of values matching the class/type of the specialist. We normalize this score between 0 and 1 and generate a ranked list of classes for each column. Every specialist type is manually mapped to one class from the UCO ontology. The ranked list of classes also include a special 'NA' (No Annotation) class for columns which do not match any of the specialist types. The score for the NA class is computed using the idea proposed in the original framework [16]. The likelihood that column should be mapped to NA is simply the product of the negative probabilities for the column to belong to each class. The Decode module can easily be extended by including additional specialists. For instance, if a system administrator decides to have a specialist for server names, a dictionary based specialist can easily added without affecting the rest of the module.

C. Relationship Generator

The *Relationship Generator* module uses the top ranked class for each column from the previous module to identify and map every pair of column in the log file to a relationship from the extended UCO ontology. The columns mapped to NA are not considered in this module. We use RDFLib[4], a python library to parse and query the ontology. For every pair of column, the module uses the top ranked classes and queries the UCO ontology to identify properties/relations for which the classes either act as the domain of the property or range of the property. For instance, we query and identify all properties where Class C_1 for column 1 is the domain and Class C_2 for column 2 is the range and vice–versa to ensure an exhaustive search of the ontology. We generate a set of multiple relation for cases where more than one match is found.

D. Generate RDF

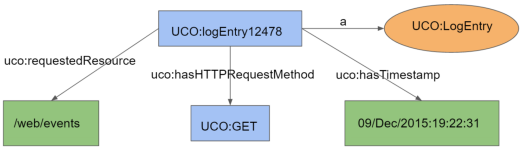

Figure 3. Partial RDF representation of the data from Figure 1

The *Generate RDF* module generates a RDF Linked Data representation of the log file content using the inferred semantic classes and relations from the UCO ontology. Figure 3 shows a partial visual representation of the RDF Linked Data for the log file from Figure 1 Every row of a log file is represented as an instance of the *LogEntry* class. The relations between the log file columns are used to represent the relation between the data elements in a log entry. For example, consider the last row from the sample log file in Figure 1. The row is represented by the instance *logEntry12478* which is of type *LogEntry*. The log

[4]https://github.com/RDFLib/rdflib

entry has data elements which are values of the properties *hasHTTPRequestMethod* and *requestedResource*. The log entry has a associated timestamp which is captured by the *hasTimestamp* property. This RDF representation can be stored in a knowledge base for further integration with data from other sources such as Intrusion Detection Systems and reasoning to detect and prevent attacks.

IV. Evaluation

A. Data Setup

We evaluate our framework using two different datasets of log files viz., original log files and synthetically generated log files. The original dataset contains actual log files obtained from various tools and services running at the operating system level such as syslog, kern, auth, etc. We also obtain log files from commonly used services like apache2, mongodb, sendmail and printer logs.

The synthetic dataset includes artificially generated log files using known columns from the original log files. The synthetic log files were generated to test the framework against log files with unknown source and format. We generate each synthetic file by randomly choosing column headers (types) from the original set. The numbers of columns to be included in each was also chosen randomly. The values added in each column were generated at random too to ensure variation in length and other characteristics of the column. To simulate the unpredictable nature of the log files, limited amount of noise was also introduced in the values in each column.

Table 4 gives a general overview about the log files in each dataset.

Dataset	*Tables*	*# of Cols.*	*Avg. Cols./ Table*	*Avg. Rels/ Table*
Original	11	78	7	3
Synthetic	20	150	8	5

Figure 4. Number of log files, total columns present, average columns and relations per table in each dataset

B. Tabulation

We first evaluate both the precision and recall of the splitting algorithm in the Tabulate module. We manually annotate each log file in both the datasets with the expected number of columns and exact column boundaries i.e separation of text between each column. An additional pre-processing step was performed on the dataset by eliminating the long trailing text often found at the end of each log entry. In practice, we can find several tools which can help clean log files and strip unwanted text.

The log files in the both the datasets were processed by the Tabulate module; we counted the number columns produced in each file and also compared the column boundaries with those in the annotated log files. A column was considered to be correctly identified only if the boundaries matched. Figure 5 presents precision and recall for the splitting algorithm in the Tabulate module on both the original and the synthetic datasets and also it's comparison with the Splunk.

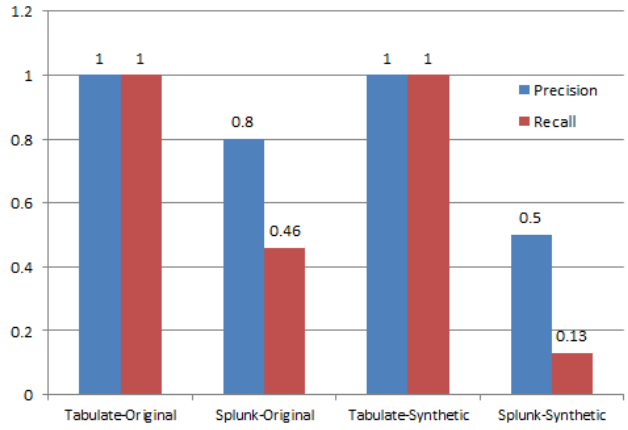

Figure 5. Precision and recall for column separation

Tabulate's precision and recall over both the datasets is very good. In comparison, Splunk obtains a precision of 0.8 and a recall of 0.46 over the original dataset. Splunk is able to give good performance over the orignal dataset as it contains log files with known format and from known sources. However, when it comes to log files from unknown sources or unknown formats which is the nature of log files in the synthetic dataset, Splunk's performance is poor with a precision and recall of 0.5 and 0.13 respectively.

We also evaluated our Tabulate module without eliminating the trailing text at the end. Tabulate is able to get a precision of 0.45 and a recall of 0.89 over the original dataset, whereas a precision of 0.55 and 0.87 over the synthetic dataset. The low precision for both the datasets can be attributed to columns with values spanning multiple words without any surrounding characters (such as quotes). The Tabulate module split the descriptive column with large text, often found as the trailing entry for each line, into several individual columns which lead to extraneous columns and thus a lower precision.

The primary goal of the Tabulate module though is to identify all the necessary columns, even if it does so at the expense of generating few incorrect ones. The high recall for both the datasets shows that Tabulate is successfully able to identify all the expected and relevant columns. The Decode module can act a filter for the extraneous columns with junk values as they would get annotated as NA.

C. Identifying Column Types

The split log files from both the datasets are annotated using the Decode module which generated a ranked list of classes for each column. We presented this ranked list for

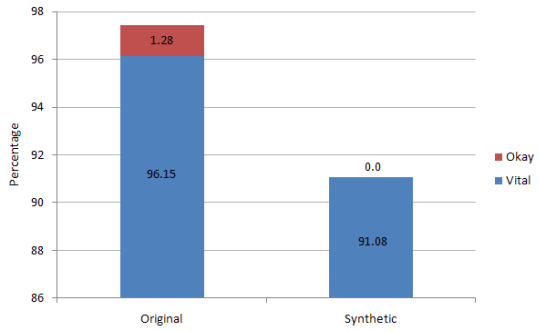

Figure 6. Percentage of vital and okay classes at rank 1

every column along with the log file to a set of human annotators. Each annotator marked every class as either *vital*, *okay* or *incorrect*. For instance, an annotator could mark, the *URL* class as *vital*, *FilePath* as *okay* and *HTTPMethod* as *incorrect*, for the column *URL*. Each column may have multiple *vital*, *okay* or *incorrect* labels. Figure 6 presents the number of vital and okay classes at rank 1 for both the datasets. 97.43% of the classes at rank 1 were found to be relevant (i.e. vital or okay) for the original dataset, whereas only 91.08% of the classes at rank 1 were considered relevant in the synthetic dataset.

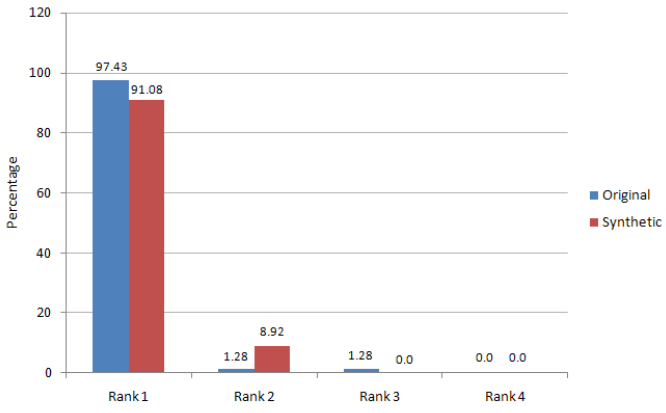

Figure 7. Percentage of relevant classes at ranks 1 to 4

Figure 7 shows the distribution of relevant classes at different ranks. Most relevant classes are found at rank 1; very few relevant classes are found at ranks 2 and 3. None of the lower ranks from 4 include any relevant classes. The distribution of relevant classes for both the datasets are comparable. It can be seen that the percentage of relevant classes at rank 1 are lower for Synthetic dataset because of instances such as misclassification of *URL* as *FilePath*. Our synthetically generated dataset consisted of more misclassification instances thus leading to slightly lower relevant classes at the top rank.

D. Identifying Column Relations

The *Relationship Generator* module generated a list of possible relations for each column pair using the UCO ontology. A manually selected set of 5 relevant relations was presented to human evaluators who again marked each relations as either *vital*, *okay* or *incorrect*. The top relation from each set was used to compute precision and recall. The top relation was considered a correct prediction if it was either *vital* or *okay*. Figure 8 shows the precision and recall calculated for the Relationship Generator module. The lower precision and recall in the synthetic dataset can be attributed to several false positive predictions.

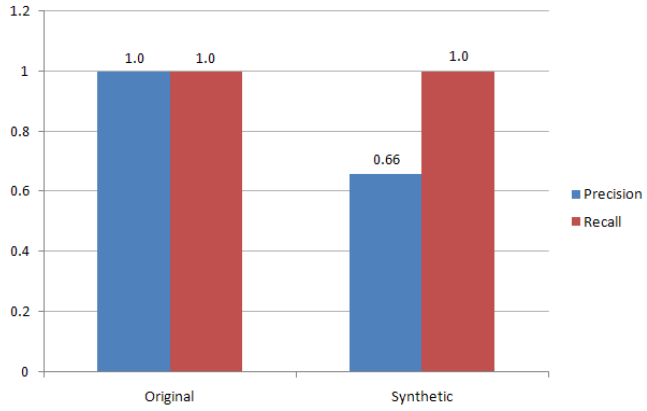

Figure 8. Precision and recall for relation annotations

Figure 9 shows the distribution between vital and okay labels for the top relation in the set. All the top relations in the original dataset were considered vital by our human annotators, whereas 93.33% were considered vital in the synthetic dataset. In comparison, tools such as Splunk do not have advanced capabilities such as identifying semantic relations between columns in a log file or generating a machine understandable representation of log file data.

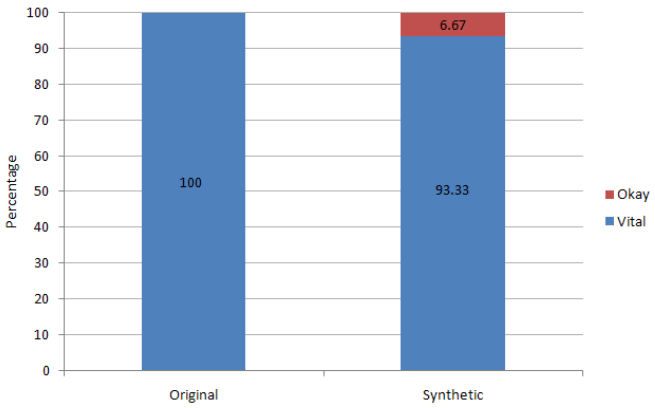

Figure 9. Percentage of vital and okay relation labels

V. Conclusion and Future Work

In this paper we described a framework that can process any log file with unknown source and format, and automatically generate a semantic interpretation of both its scema and contents in the form of RDF linked data triples. We develop a splitting algorithm which can successfully generate a table like structure for log files. We also extend existing techniques to successfully annotate every column in a log file with a semantic class and map selected pairs of column to relation from the given ontology. Preliminary evaluations show promising results. In future work we plan to extend the UCO ontology to incorporate additional classes and relations to better suit this domain. We would also like to process the verbose textual description which form the trailing entry of logs and extract concepts related to security vulnerabilities and threats.

Acknowledgment

Support for this work was provided by NSF grants 1250627, 1228198 and a gift from Microsoft. One of the authors also acknowledges support from the Oros Family Professorship endowment.

References

[1] K. Kent and M. Souppaya, "Guide to computer security log management – recommendations of the national institute of standards and technology," NIST, 2006.

[2] "Analysis of the siem and log management market," Frost and Sullivan, 2013.

[3] S. More, M. Matthews, A. Joshi, and T. Finin, "A knowledge-based approach to intrusion detection modeling," in *IEEE Symposium on Security and Privacy Workshops*. IEEE, 2012, pp. 75–81.

[4] G. Klyne and J. J. E. Carroll, "Resource description framework (rdf): Concepts and abstract syntax," World Wide Web Consortium, Tech. Rep., February 2004.

[5] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[6] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pp. 205–227, 2009.

[7] A. Joshi, R. Lal, T. Finin, and A. Joshi, "Extracting cybersecurity related linked data from text," in *Seventh International Conference on Semantic Computing (ICSC)*. IEEE, 2013, pp. 252–259.

[8] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan, "Extracting information about security vulnerabilities from web text," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 3. IEEE, 2011, pp. 257–260.

[9] C. H. do Nascimento, R. E. Assad, B. F. Lóscio, and S. R. L. Meira, "Ontolog: A security log analyses tool using web semantic and ontology," in *2nd OWASP Ibero-American Web Applications Security Conference*, 2010, pp. 1–12.

[10] "Splunk," http://www.splunk.com, accessed: 2015-07-01.

[11] V. Mulwad, T. Finin, and A. Joshi, "A Domain Independent Framework for Extracting Linked Semantic Data from Tables," in *Search Computing - Broadening Web Search*. Springer, July 2012, pp. 16–33, lNCS volume 7538.

[12] V.Mulwad, T.Finin, and A.Joshi, "Semantic message passing for generating linked data from tables," in *Proceedings of the 12th International Semantic Web Conference*. Springer, October 2013.

[13] V. Mulwad, "TABEL - A Domain Independent and Extensible Framework for Inferring the Semantics of Tables," Ph.D. dissertation, University of Maryland, Baltimore County, January 2015.

[14] Z. Syed, A. Padia, M. L. Mathews, T. Finin, and A. Joshi, "UCO: A Unified Cybersecurity Ontology," in *AAAI Workshop on Artificial Intelligence for Cyber Security*. AAAI Press, February 2016.

[15] J. Undercofer, A. Joshi, T. Finin, and J. Pinkston, "Using DAML+ OIL to classify intrusive behaviours," *Knowledge Engineering Review*, vol. 18, no. 3, pp. 221–241, January 2004.

[16] N. Puranik, "A specialist approach for the classification of column data," Master's thesis, University of Maryland, Baltimore County, 2012.