Discovering Insider Threats from Log Data with High-Performance Bioinformatics Tools

Markus Wurzenberger, Florian, Skopik, Roman Fiedler
Austrian Institute of Technology, Digital Saftey and Security Department
Donau-City-Strasse 1
1220 Vienna, Austria
firstname.lastname@ait.ac.at

Wolfgang Kastner
Vienna University of Technology, Institute of Computer Aided Automation
Treitlstrasse 3
1040 Vienna, Austria
k@auto.tuwien.ac.at

ABSTRACT

Since the number of cyber attacks by insider threats and the damage caused by them has been increasing over the last years, organizations are in need for specific security solutions to counter these threats. To limit the damage caused by insider threats, the timely detection of erratic system behavior and malicious activities is of primary importance. We observed a major paradigm shift towards anomaly-focused detection mechanisms, which try to establish a baseline of system behavior – based on system logging data – and report any deviations from this baseline. While these approaches are promising, they usually have to cope with scalability issues. As the amount of log data generated during IT operations is exponentially growing, high-performance security solutions are required that can handle this huge amount of data in real time. In this paper, we demonstrate how high-performance bioinformatics tools can be leveraged to tackle this issue, and we demonstrate their application to log data for outlier detection, to timely detect anomalous system behavior that points to insider attacks.

CCS Concepts

•**Security and privacy** → **Intrusion detection systems;** Systems security; Network security; •**Information systems** → *Data mining;*

Keywords

log data clustering; anomaly detection; outlier detection

1. INTRODUCTION

Many of today's ICT security solutions promise an automatic detection (and even mitigation) of malicious system behavior. They apply complex detection schemes and heuristics, yet the financial loss caused by cyber attacks is inexorably increasing. In 2014, the financial damage caused by insider attacks was estimated to 575 billion dollars[1]. An insider threat can either be an employee of the organization or an outside person who pretends to be an employee by using stolen or false credentials. Most insider threats are employees or ex-employees, who either believe that they have been mistreated by their company and want to take revenge, plan to sell intellectual properties, or were involved in illegal activities and seek to hide traces by manipulating internal data (e.g., logs of financial transactions). Insider threats usually result in data ex-filtration (theft of intellectual property) and deletion, financial loss or sabotage. Effectively combating insider threats requires deploying data-centric instead of system centric security, data encryption and data access logging, as well as establishing centralized logging, which allows determining a baseline of normal system behavior, and finally using intrusion detection/prevention systems (IDS/IPS), SIEMs, outlier detection and other log analysis tools for detecting threats.

1.1 Insider Threat Scenario

Figure 1 outlines a company's network environment, consisting of client machines and company internal services. Employees can connect to the internal services only through the firewall. For example a whitelist can prevent connections from external devices using an unknown IP address and MAC address (if we neglect spoofing attacks). However some employees need direct access to the database. If one considers a web-based customer relationship management (CRM) portal, for example common sales employees only have access to the database through the web servers, which allows them only to access information related to their specific working area and not to access confidential information such as banking accounts; the sales manager instead has direct access to the database server and to all information stored there. Figure 1 describes the following two possible scenarios of an attack by an insider threat:

(i) The insider attacker starts ex-filtrating the database, which heavily raises the number of logged queries related to the database server. This data ex-filtration would not violate any firewall rule and also the access to the database would not be suspicious; only the frequency of accesses would be anomalous. While it is hard to design a rigid rule-based schema to prevent

MIST'16, October 28 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2995959.2995973

[1] http://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf

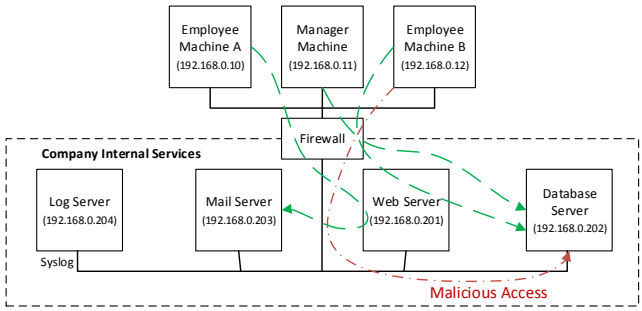

Figure 1: Description of an insider attack. The dashed red line represents a malicious database access, the dashed green lines represents normal accesses.

such attacks, log line clustering and applying a time series analysis allow to detect a change in the way the database server is utilized, and thus the attack.

(ii) The second scenario can be either caused by a misconfiguration or by information leakage. *Misconfiguration* – the port the manager uses to directly connect to the database is open for every employee and an insider threat only needs to find out the port number to connect directly to the database. *Information leakage* – The attacker found out the administrator credentials to get direct access to the database server. In both cases, the insider attacker uses the malicious access path (marked by the dashed red arrow in Fig. 1) to connect to the database server. This insider threat attack can be detected by clustering for outlier detection. There will exist no cluster describing a direct access to the database server with the IP address and MAC address used by employee's machine B.

1.2 Application of Bioinformatics Tools

Clustering techniques are very effective tools for periodically reviewing rare events (outliers) and checking frequent events by comparing cluster sizes over time (e.g., trends in the number of requests to certain resources). Furthermore, a methodology and supporting tools to review log data and to find anomalous events in log data are needed. Existing solutions are suitable to cover all these requirements, but they still show crucial deficits. Most of them, such as SLCT [4], implement word-based matching of log entries, but do not identify synonyms which only differ in one character, such as 'php-admin' and 'phpadmin', or identify similar URLs as completely different words. Hence, the implementation of character-based matching with comparable speed such as word-based matching is necessary. Furthermore, existing tools are often not useable for processing large log files to extract cluster candidates over months and to perform gradual logging, which can be applied to generate a base corpora to identify how and where log clusters are changing over time.

In the domain of bioinformatics, various methods have been developed to analyze and study the similarity of biologic sequences (DNA, RNA, amino acids), group similar sequences and extract common properties. The algorithms to implement these features need to fulfill some strict requirements: (i) adequate digital representation, (ii) dealing with natural variations, (iii) dealing with artificial inaccuracies, (iv) dealing with massive data volumes.

In general, all these requirements also apply to modern log data processing as (i) data needs to be processed extremely fast (this means depending on the application approximately in real time); (ii) data analysis needs to be scheduled in parallel in order to scale; and (iii) the process needs to accept certain inaccuracies and errors that occur due to conversion errors from varying character encodings, and slight differences in configurations and output across software versions. Furthermore, these tools aim at processing character sequences without taking into account their semantics.

As a consequence, if the mentioned tools are not applied to biologic sequences but to re-coded (converted) digital sequences, such as log data (or even malware code), all of the peculiar properties of algorithms can be harnessed directly, without the need to design and implement complex tools again. Sequence alignment algorithms that compare two amino acid sequences are for example the Needleman-Wunsch algorithm and the Smith-Waterman algorithm [3]. Algorithms for clustering amino acids that exploit sequence alignment are CD-HIT, CLUSTAL, UCLUST and DNA-CLUST [1].

In this paper, we define a method for re-coding log data into the alphabet used for representing canonical amino acid sequences, which enables the application of high-performance bioinformatics tools for outlier detection in the domain of log data processing. The proposed model implements a self-learning white-listing approach, that does not account for any knowledge about syntax and semantics of the processed log data,

The remainder of the paper is structured as follows. Section 2 describes the theoretical model of our approach for discovering outliers and Sect. 3 evaluates our approach. Finally, Sect. 4 concludes the paper and discusses future work.

2. MODEL FOR APPLYING BIOINFORMATICS CLUSTERING TOOLS ON LOG DATA

The following section defines the theoretical model for applying high-performance bioinformatics tools for clustering computer log data. The proposed modular model comprises several steps from re-coding log data to the alphabet used for describing amino acid sequences to interpretation and analysis of the output for cyber security application: (i) re-code and format log data, (ii) compare pairs of log lines according to their degree of similarity, (iii) cluster log lines, (iv) re-translate data, (v) detect outliers.

2.1 Re-coding Model

Log data from ICT systems is usually modeled in human-readable textual form. Therefore, before tools from the domain of bioinformatics can be applied to it, step (i) has to be carried out, i.e., re-coding the log data using the alphabet used for representing amino acids and converting it into a format, which can be exploited by the applied tools.

A basic unit of logging information, e.g., one line for line-based logging, or one XML-element, is called a textual log atom L_{text} which consists of a series of symbols s – typically letters and numbers (Eq. 1).The used alphabet to represent log data consists (in most cases) of UTF-8 encoded characters (256 different symbols). In the following, A_{UTF-8} refers to this alphabet.

$$L_{text} = \langle s_1 s_2 s_3 \dots s_n \rangle \text{ where } s_i \in A_{UTF-8} \quad (1)$$

However, data represented in this format is unsuitable as input to bioinformatics tools. Such tools require input (biologic sequences) encoded with symbols of the alphabet A_{bio} defined for amino acid or DNA sequences. This alphabet consists of 20 symbols only, which represent the 20 canonical amino acids.

A re-coding function takes an input stream encoded as UTF-8 data and transforms it into a representation L_{bio} (Eq. 2) that is processable by bioinformatics tools.

$$L_{bio} = \langle s_1 s_2 s_3 \dots s_m \rangle \text{ where } s_j \in A_{bio} \quad (2)$$

In the simplest case, this transformation is a straightforward bijective mapping, where one A_{UTF-8} symbol is represented by two symbols from A_{bio}. However, for data where certain larger blocks frequently appear, those whole blocks (e.g., server names or IP addresses) could be replaced with a single symbol. This would effectively allow compression of data. Even further information loss could be – depending on the application – acceptable. For instance, frequently appearing symbol blocks could be replaced by applying a more intelligent, but one-way mapping, e.g., not a whole IP address but just the last byte or the address' cross sum could be translated to A_{bio}. Another example are paths (from Web server logs), where each component of a path could be translated through hashing into single symbols of A_{bio}. Furthermore, symbols can be grouped by type, so that all separators such as '/', ';' or spaces can be replaced by one specific element of A_{bio}. Finally, more symbols could be spent on the variable parts of log lines (those with higher entropy) and less symbols (or no symbols at all) on the rather static parts.

One simple - but effective - method for re-coding log data into A_{bio} is to convert symbol by symbol each $s_i \in L_{text}$ into two corresponding $s_j \in L_{bio}$ symbol by symbol (without any loss of information). For this purpose, each symbol in L_{text} (i.e., the single letters of the words in a log line) is converted to its numerical representation in UTF-8. The result of this operation is L_{utf} (Eq. 3).

$$L_{utf} = \langle a_1, a_2, a_3 \dots a_n \rangle \text{ where } a_i \in \{0, \dots, 255\} \quad (3)$$

In a second step, each numerical value $a_i \in L_{utf}$ is converted into two symbols of the alphabet A_{bio}. Since the size of this alphabet is always 20, a straightforward solution (which uses the whole possible input range) is to divide each $a_i \in L_{utf}$ by 20, and additionally keep the rest of this division. Eventually, both results s_1 (the result of the integer division) and s_2 (the rest of the division) are mapped via a simple conversion table to A_{bio}. Concatenating all these symbols in a single stream effectively produces L_{bio} – the input to alignment and clustering tools from the domain of bioinformatics.

The re-coding process is further described in Alg. 1. Function `utf2num` looks up the decimal symbol number in a standard UTF-8 table (e.g., the letter 'A' corresponds to the number 65). The function `num2bio` looks up the letter representation of the numbers 0 to 19.

A simple option to reduce/compress the amount of data needed to represent one log line by 50% is to omit the leading

Algorithm 1 Re-Coding L_{text} into L_{bio}

```
1: L_bio ← ∅
2: L_utf ← ∅
3: for all s_i ∈ L_text do
4:     L_utf ← L_utf ⊕ utf2num(s_i)
5: end for
6: for all a_i ∈ L_utf do
7:     s_1 ← a_i/20
8:     s_2 ← a_i%20
9:     L_bio ← L_bio ⊕ num2bio(s_1) ⊕ num2bio(s_2)
10: end for
```

character s_1, instead of representing each $s_i \in L_{text}$ by two $s_j \in L_{bio}$ (cf. Alg. 1). This character – s_1 – has lower entropy compared to the trailing s_2 (cf. Alg. 1). As a result the length of L_{bio} can effectively be halved by accepting a "small" ambiguity. In the following, we always apply this method for recording L_{text} into L_{bio}.

To complete step (i), the data has to be transformed into the correct format. Most of the bioinformatic tools require data in the FASTA format[2]. The FASTA format requires a header, which can be used to store information for the re-translation implemented by step (iv).

2.2 Comparing and Clustering Log Data

In this section we describe how log lines re-coded into L_{bio} can be compared by applying sequence alignment algorithms and how they then can be clustered based on their similarity.

2.2.1 Pairwise Log Line Comparison

Sequence alignment algorithms, which are applied in step (ii) to compare two log lines, form the base for most bioclustering tools. Alignment algorithms use a scoring function d to calculate the distance between two sequences. When comparing two sequences L^A_{bio} and L^B_{bio} element by element, three possible cases can occur:*mismatch* (symbol s^A_j was replaced by symbol s^B_j), *deletion* (symbol s^A_j was removed in L^B_{bio}, *insertion* (symbol s^B_j was inserted in L^B_{bio}).

The alignment between two amino acid sequences is always built under the assumption that L^A_{bio} and L^B_{bio} have common ancestors, i.e., they are homologous. This means in the end the alignment which refers to the highest similarity is chosen. A similarity score specifies how similar two amino acid sequences are. The predefined score for a match is usually constant. In most cases, the score for a mismatch depends on the probability that s^A_j can evolve to s^B_j over time. The score for a gap caused by deletions or insertions is also predefined and can depend on the size of the gap, or if a gap is opened or just extended. The simplest definition for a scoring function d relies on unit costs and does not take into account that s^A_j could evolve to s^B_j by a specific probability:

$$d(s^A_j, s^B_j) = \begin{cases} 1 & \text{if } s^A_j = s^B_j, \\ -1, & \text{is } s^A_j \neq s^B_j, \end{cases} \quad (4)$$

and -1 for deletions and insertions. When comparing two amino acid sequences, there are usually various options to build the alignment (c.f. Tab. 1). In our model, since the sequences considered as homologous, the alignment with the highest score is chosen, because a higher score suggests a higher similarity.

[2]https://blast.ncbi.nlm.nih.gov/Blast.cgi

option	alignment	score	similarity
(i)	GAC GC-	$1-1-1=-1$	33.33%
(ii)	GAC- --GC	$-1-1-1-1-1=-5$	0%
(iii)	GAC G-C	$1-1+1=1$	66.66%

Table 1: Example alignments

In the proposed model, the similarity, between the two amino acid sequences can be calculated as the ratio between the number of identical symbols in the alignment and the length of the alignment as shown in Eq. (5). Equation (5) is a normalized version of the inverted Lvenshtein distance, i.e., the identical symbols are calculated instead of the number of changes. The third column of Tab. 1 shows the similarities based on Eq. (5).

$$similarity = \frac{identicalSymbolsAlign(L^A_{bio}, L^B_{bio})}{lengthOfAlign(L^A_{bio}, L^B_{bio})} \quad (5)$$

2.2.2 Log Line Clustering

Step (iii) performs clustering log data and is based on the previously defined alignment of two bio-encoded log lines. By re-coding a whole log data set and subsequent pairwise comparison of bio-encoded log lines through sequence alignment, distances can be determined by calculating the similarity of two sequences (cf. Eq. (5)). Clustering tools then cluster the bio-encoded sequences so that the distances between any two cluster members $c_i \in C$, $c_j \in C$ is lower than the distance to the next cluster center. This analysis can be performed with various existing bio-clustering tools, such as CD-HIT[3].

For further analysis of the clustering output, the sequences have to be re-translated into understandable text – this is done in step (iv). The FASTA format provides the possibility to store the position of a log line in the original log file in the header. Using this information, it is possible to look up the corresponding log line for each bio-encoded sequence in the input log file.

2.3 Outlier Detection

The following section deals with step (v) – outlier detection for detecting anomalies [2]. Outlier detection aims at identifying so called point anomalies. These outliers are clusters including just a small amount of elements and/or usually a large distance to other clusters, which define the normal state of an network environment. In case of log data, outlier clusters include rare or atypically structured events (log entries). Those outliers are log entries that require further investigations. Eventually, the previously defined model allows to apply high-performance tools from the domain of bioinformatics on log data to cluster log lines. During the re-translation from A_{bio} to A_{UTF-8} the clusters can be sorted by their size to detect clusters of small size, which represent outliers.

3. EVALUATION

To test the proposed approach we implemented a prototype which applies CD-HIT for clustering. Furthermore we generated a semi-synthetic dataset simulating the insider threat (ii) described in Sect. 1.1. We tested both proposed methods for re-coding and could detect the generated outliers with a similarity threshold of 92%, when re-coding without information loss, and 88% when re-coding wih loss of information. As a result the false positive rate for the second method was slightly lower. Furthermore, we observed that the runtime of the algorithm scales linearly with the number of processed log lines.

[3]http://weizhongli-lab.org/cd-hit/

4. CONCLUSION AND FUTURE WORK

This paper describes a novel model, which allows to apply high-performance bioinformatics tools in the context of anomaly detection on log data produced in computer networks. Since most of the bioinformatics tools operate on canonical amino acid sequences, we introduced two different methods to re-code log data coded in UTF-8 code (255 symbols), to the alphabet of canonical amino acids (20 symbols). We further demonstrated how the re-coded log data can be clustered applying bioinformatics algorithms and tools for generating sequence alignments, and we described how the output can be used for outlier detection to reveal anomalous system behavior.

In the future, we plan to focus on further development of the re-coding model described in Sect. 2.1 and the scoring system defined in Sect. 2.2.1. We intend to modify the re-coding function, so that often re-occurring parts of log files (e.g., static texts) are translated into less symbols and therefore accept a higher loss of information for these parts; less frequent parts, which include the more interesting variable parts (e.g., IP addresses, user names, port numbers) of log lines, should be translated without loss of information. We also plan to investigate if the scoring system can be adjusted so that similarly to the analysis of amino acid sequences, the penalty score is lower for highly related letters and higher in otherwise. Moreover, we aim to present a detailed evaluation of the proposed model to demonstrate the detection capability, the accuracy and precision, as well as the efficiency of our approach. Moreover, we want to compare our approach with already existing methods. We finally intend to investigate how our approach can be used for time series analysis to detect different attack patterns.

Acknowledgments

This work was partly funded by the European Union FP7 project ECOSSIAN (607577) and carried out in course of a PhD thesis at the Vienna University of Technology funded by the FFG project BAESE (852301).

5. REFERENCES

[1] M. Ghodsi, B. Liu, and M. Pop. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics*, 12(1):1, 2011.

[2] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[3] D. W. Mount. Sequence and genome analysis. *Bioinformatics: Cold Spring Harbour Laboratory Press: Cold Spring Harbour*, 2, 2004.

[4] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *IPOM*, pages 119–126, Oct 2003.