Anomaly Detection from Log Files Using Data Mining Techniques

Jakub Breier[1,2] and Jana Branišová[3]

[1]Physical Analysis and Cryptographic Engineering, Temasek Laboratories@NTU
[2]School of Physical and Mathematical Sciences, Division of Mathematical Sciences, Nanyang Technological University, Singapore
jbreier@ntu.edu.sg
[3]Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava, Slovakia
branisovaj@gmail.com

Abstract. Log files are created by devices or systems in order to provide information about processes or actions that were performed. Detailed inspection of security logs can reveal potential security breaches and it can show us system weaknesses.
In our work we propose a novel anomaly-based detection approach based on data mining techniques for log analysis. Our approach uses Apache Hadoop technique to enable processing of large data sets in a parallel way. Dynamic rule creation enables us to detect new types of breaches without further human intervention. Overall error rates of our method are below 10%.

Keywords: Log Analysis, Anomaly Detection, Data Mining, Apache Hadoop, MapReduce

1 Introduction

A system or a device can provide information about its state in the form of log files. These files contain information if it works properly or which actions or services have been executed. By analyzing such information we can detect anomalies which can reveal potential security breaches. Since complex systems provide huge amounts of log records, it is not feasible to analyze them manually, therefore it is necessary to use automatized methods for this purpose. Outcomes of this process can help with a correct setup of network devices, it is possible to reveal non-privileged access to the system or even to find a person who performed the breach [8].

There are various types of logging. *Security logging* encompasses obtaining information from security systems and can be used to reveal potential breaches, malicious programs, information thefts and to check the state of security controls. These logs also include *access logs*, which contain data about user authentication. *Operational logging* reveals information about system errors and malfunctions, it is useful for a system administrator that needs to know about current system state. *Compliance logging* provides information about compliance with security requirements and it is sometimes similar to security logging. There are two subtypes of these logs: logging of security of information systems with respect to data transfer and storage, e.g. PCI DSS or HIPAA standard compliance, and logging of system settings.

In our work we focus on anomaly detection in log files with the help of data mining techniques. By comparing different approaches of anomaly and breach detection we decided to use a method based on dynamic rule creation with the data mining support. The main advantage of this method is its ability to reveal new types of breaches and it minimizes the need of manual intervention. We have also investigated possibilities for more effective log analysis of large data volumes because of the size and amount of log files, which has increasing tendency. By implementing the Apache Hadoop technology we created a single node cluster for parallel log processing using the *MapReduce* method. Time needed for log analysis has greatly increased and the algorithm implemented in Hadoop was able to process data faster than the standard algorithm using tree-based structure. Also, by adding more computing nodes it is easy to improve processing time, if necessary.

The rest of the paper is organized as follows. Section 2 provides an overview of related work in this area. Section 3 describes methods that could be used for anomaly detection. In section 4 we present the design of our solution, and finally, section 6 concludes this paper.

2 Related Work

There are several works proposing usage of data mining methods in log file analysis process or in detecting security threats in general.

Schultz et al. [6] proposed a method for detecting malicious executables using data mining algorithms. They have used several standard data mining techniques in order to detect previously undetectable malicious executables. They found out that some methods, like Multi-Label Naive Bayes Classification can achieve very good results in comparison to standard signature-based methods.

Grace, Maheswari and Nagamalai [4] used data mining methods for analyzing web log files, for the purposes of getting more information about users. In their work they described log file formats, types and contents and provided an overview of web usage mining processes.

Frei and Rennhard [2] used a different approach to search for anomalies in log files. They created the Histogram Matrix, a log file visualization technique that helps security administrators to spot anomalies. Their approach works on every textual log file. It is based on a fact that human brain is efficient in detecting patterns when inspecting images, so the log file is visualized in a form that it is possible to observe deviations from normal behavior.

Fu et al. [3] proposed a technique for anomaly detection in unstructured system logs that does not require any application specific knowledge. They also included a method to extract log keys from free text messages. Their false positive rate using Hadoop was around 13% and using SILK around 24%.

Makanju, Zincir-Heywood and Milios [5] proposed a hybrid log alert detection scheme, using both anomaly and signature-based detection methods.

3 Detection Methods

According to Siddiqui [7], there are three main detection methods that are used for monitoring malicious activities: scanning, activity monitoring and integrity check. *Scanning* is the most widely used detection method, based on searching for pre-defined strings in files. Advanced version of scanning includes heuristic scanning which searches for unusual commands or instructions in a program. *Activity monitoring* simply monitors a file execution and observes its behavior. Usually, APIs, system calls and hybrid data sources are monitored. Finally, *integrity checking* creates a cryptographic checksum for chosen files and periodically checks for integrity changes.

Data mining is relatively new approach for detecting malicious actions in the system. It uses statistical and machine learning algorithms on a set of features derived from standard and non-standard behavior. It consists of two phases: data collection and application of detection method on collected data. These two phases sometimes overlap in a way that selection of particular detection method can affect a data collection.

Data can be analyzed either statically or dynamically. Dynamic analysis observes a program or a system during the execution, therefore it is precise but time consuming. Static analysis uses reverse-engineering techniques. It determines behavior by observing program structure, functionality or types of operation. This technique is faster and it does not need as much computational power as dynamic analysis, but we get only approximation of reality. We can also use hybrid analysis - first, a static analysis is used and if it does not achieve correct results, dynamic analysis is applied as well.

3.1 Parallel Processing

A huge amount of log data is generated every day and it is presumed that this amount will grow over time. Therefore, it is necessary to improve the process of the log analysis and make it more effective. We have chosen Apache Hadoop[1] technology for this purpose with the *MapReduce* [1] programming model. *MapReduce* is used for processing large data sets by using two functions. *Map* function processes the data and generates a list in the key-value form. *Reduce* function can be then used by the user for joining all the values with the same key. Hadoop architecture is based on distributing the data on every node in the system. The resulting model is simple, because MapReduce handles the parallelism, so the user does not have to take care about load balancing, network performance or

[1] http://hadoop.apache.org/

fault tolerance. Hadoop Distributed File System can then effectively store and use the data on multiple nodes.

4 Design

The main idea for the design of our solution is to minimize false positives and false negatives and to make the anomaly identification process faster.

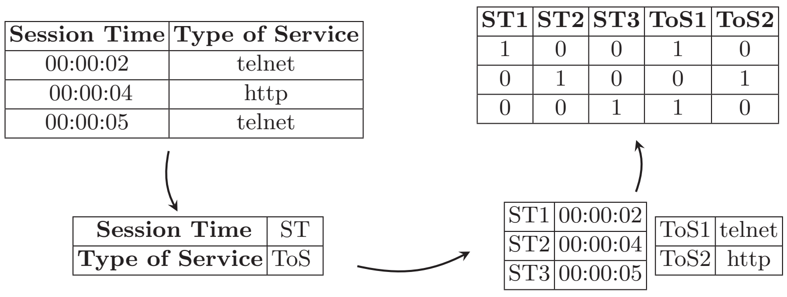
The steps of the algorithm are following. First, a testing phase is performed and rules are made from the testing data set. The outcome of this phase is an anomaly profile that will be used to detect anomalies in network devices log files. For creating rules, log file is divided into blocks instead of rows. A block is identified by the starting time, session duration and type of a service. We will use a term 'transaction' for particular block. This approach allows us to create a rule based on several log files from different devices or systems, so that one transaction can contain information from various sources. For creating uniform data sets, which can be processed by different algorithms, each transaction is transformed in a binary string form. For a spatial recognition of a log record in transaction, each record in the original log file will be given a new attribute - transaction ID.

For the detection program it is necessary to be able to process various log file formats, therefore we decided to use configuration files which will help to determine each attribute position.

4.1 Data Transformation

Data transformation includes creation of a new data set that contains the binary data only. The advantage of such data representation is ability to process it with various algorithms for association rules creation. Example of such a transformation is depicted in Table 1.

Table 1. Binary Transformation Example.



Session Time	Type of Service
00:00:02	telnet
00:00:04	http
00:00:05	telnet

Session Time	ST
Type of Service	ToS

ST1	00:00:02
ST2	00:00:04
ST3	00:00:05

ToS1	telnet
ToS2	http

ST1	ST2	ST3	ToS1	ToS2
1	0	0	1	0
0	1	0	0	1
0	0	1	1	0

To avoid a problem with large dimension number by using binary representation of log records, we propose a data reduction. This reduction is achieved by inserting values into categories and using an interval representative instead of a scalar or time value. Binary string contains a numerical value of 1 for values which are present in the record and a numerical value of 0 otherwise. However, some of the values are unable to reduce, such as IP addresses or ports.

4.2 Transaction and Rule Creation

Transaction and rule creation algorithm works as follows. It loads each record from log files line by line and stores them in the same block if they were created in the same time division, within the same session and if the IP addresses and ports are identical. If they can be identified as related, a transaction is created. Then it is decided whether this transaction fulfills conditions to be included in the anomaly profile. If yes, a new rule is created, if no, this transaction is ignored for the further rule creation process.

A rule contains attributes in a binary form that are defined in configuration file. It always contains some basic attributes related to time and session parameters and also a device ID, from which particular log record originates.

The anomaly finding algorithm first loads a set of previously created rules from the database. Then it sequentially processes the log files intended for analysis and creates transactions from these files. This transaction is then compared with the set of rules and if it is identified as an anomaly, it is stored for further observations.

4.3 Processing of Large Data Sets

As stated in Section 3.1, we decided to use Hadoop technology with MapReduce programming model to process large data sets. Hadoop enables us to easily add processing nodes of independent device types. After program starts, *JobTracker* and *TaskTracker* processes are started, which are responsible for *Job* coordination and for execution of *Map* and *Reduce* methods. *JobTracker* is a coordinator process, there exists only one instance of this process and it manages *TaskTracker* processes which are instantiated on every node. MapReduce model is depicted in Fig. 1, however in our case, only one *Reduce* method instance is used. First, a file is loaded from Hadoop Distributed File System (HDFS) and it is divided into several parts. Each *Map* method accepts data from particular part line by line. It then processes the line and stores them until a rule is created (if all the conditions are met). The rule is then further processed by the *Reduce* method, which identifies redundant rules and if the rule is unique, it is written into the HDFS.

To allow nodes to access same files in the same time, but without loading them onto the each node separately, a Hadoop library for distributed cache is used.

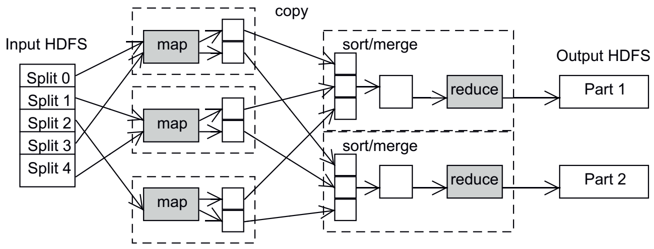


Fig. 1. MapReduce Algorithm.

5 Testing

Our anomaly detection method was implemented in Java programming language. For testing, we used Intel i7-4500U CPU with 8 GB of RAM, running Ubuntu 12.04 operating system. For testing purposes, two data sets were used: 1998 DARPA Intrusion Detection Evaluation Set[2], that was created by monitoring a system for two weeks, and Snort logs, created by analyzing the DARPA data set[3]. Snort logs contain information, if the attack was performed, or not. Based on that, we were able to determine if our anomaly detection method was able to successfully identify an intrusion, or not.

Testing was performed on a log records set of a size of 442 181 records. This set was made by merging DARPA and Snort data sets. We have split this data set into ten subsets for cross-validation purposes. In a cross-validation, a set is divided into subsets with similar sizes and each subset is used as many times as is the number of subsets. In each testing, one subset is used as a test set and the other subsets are used as training sets. For our validation, we split the main data set in a way that each subset contained log records from every day when monitoring was performed.

5.1 Data Transformation

After data sets merging, it was necessary to determine how many unique values are present in each table column. These values are stated in Table 2.

As we have already stated, high-dimensional data increases memory requirements of anomaly detection algorithm. It is possible to reduce some of the attribute values so that it can still be able to detect anomalies on a reduced set. We can analyze the 'Session Time' attribute and a process of reducing its values into intervals. These intervals are stated in Table 3.

Since the majority of records has a session time value 00:00:01, it was decided to take this value as a standalone interval. The same holds for value 00:00:02.

[2] http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/
[3] https://www.snort.org

Table 2. Occurrence of Unique Values in Merged Data Set.

Attribute	Unique Values	Min	Max
Date	10	07/20/1998	07/31/1998
Time	63 299	00:00:00	23:59:59
Session Time	884	00:00:01	17:50:36
Service	4664	n/a	n/a
Source Port	38 637	-	65 404
Destination Port	7887	-	33 442
Source IP	1 565	000.000.000.000	209.154.098.104
Destination IP	2 640	012.005.231.198	212.053.065.245
Attack Occurred	2	0	1
Attack Type	47	n/a	n/a
Alert	61	n/a	n/a

Table 3. Intervals with Highest Number of Occurences.

Session Time	00:00:01	00:00:02	(00:00:02,01:00:00>	(01:00:00,18:00:00)
Occurrences	795 421	13 873	7 987	759

Two other intervals cover longer time sessions, but since there are not many values present in each of these intervals, it was possible to make the reduction. Therefore, after reduction it was possible to change the range of values from 884 to 4 in this case, which enables significantly faster data processing.

5.2 Error Rate

Overall accuracy of the algorithm can be determined by Equation 1, where $FP =$ false positives, $FN =$ false negatives, $TP =$ true positives, $TN =$ true negatives.

$$Error\ Rate = \frac{FP + FN}{TP + TN + FP + FN} \tag{1}$$

The anomaly detection algorithm was implemented both in Java and in Hadoop. Table 4 shows values for both implementations. The table shows us that Hadoop implementation has around 1% lower error rate than Java implementation.

Table 4. Error Rate for Each Subset Using Java and Hadoop Implementations.

Error Rate	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 8	Set 9	Set 10
Java	0.095	Set 0.087	0.144	0.091	0.093	0.090	0.090	0.086	0.087	0.123
Overall	**0.098576**									
Hadoop	0.087	Set 0.077	0.142	0.083	0.083	0.083	0.080	0.077	0.077	0.121
Overall	**0.091465**									

5.3 Processing Speed

Important factor in anomaly detection is both speed of rules generation and speed of data processing. We compared our algorithm with two other anomaly detection algorithms, Apriori and FP-Growth. Apriori algorithm serves as a base for several rule-creation methods. Its disadvantage is that it needs to process the data set several times. FP-Growth algorithm uses tree-based storages for storing intermediate values. We used Weka libraries[4] for these algorithms implementations. Testing results are stated in Table 5. We can see that Hadoop implementation was the fastest among the tested algorithms. Therefore we can conclude that parallelization can bring very good results in terms of speed into the rule generation process.

Table 5. Comparison of Rule Generation Speed.

Algorithm	Java Implementation	Hadoop Implementation	Apriori	FP-Growth
Time (s)	163.1	15.6	226	93

Speed of data set processing for anomaly detection is stated in Table 6. We were comparing standard implementation in Java and implementation in Hadoop. Tests were performed on three data sets of sizes 10, 50, and 500 GB. As we can see, Hadoop can speed up this process more than ten times, even by using a single node. The Hadoop configuration was set to pseudo-distributed operation, which allowed it to run on a single-node. It is, of course, possible to add more nodes in order to improve throughput. We have tested a 10 GB data set on a three-node cluster, one node was configured as a master+slave, the other two nodes were configured as slaves only. Running time was 2040s, which gives us approximately 1.55 times better throughput than using a single-node.

Table 6. Comparison of Anomaly Detection Speed.

Data Size	10 GB	50 GB	500 GB
Number of Records (in millions)	84	423	851
Java Implementation Time (s)	32 622	164 050	330 152
Hadoop Implementation Time (s)	3 164	13 531	29 042

6 Conclusion

In our work we have proposed a way for anomaly-based breach detection from log files. We have chosen an approach based on dynamic rule creation using data

[4] http://www.cs.waikato.ac.nz/ml/weka/

mining techniques. The main advantage of such an approach is minimization of tasks requiring human interference and it is possible to detect new types of breaches.

The second goal was to reduce the time required for the log analysis, since log files are becoming larger and their number grows. We have implemented the application using Hadoop technology. We have created a single-node cluster for parallel processing, using *MapReduce* technology. This allowed us to make analysis more than ten times faster compared to using standard Java implementation. Also, it is easy to add more nodes for improving the analysis speed.

Log records were aggregated into transactions, identified by the time, session time and service type, so the manipulation with the data became more convenient. After that these transactions were transformed into binary format for faster processing.

Acknowledgement This work was supported by VEGA 1/0722/12 grant entitled "Security in distributed computer systems and mobile computer networks."

References

1. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008.
2. A Frei and M. Rennhard. Histogram matrix: Log file visualization for anomaly detection. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 610–617, March 2008.
3. Q. Fu, J.-G. Lou, Y. Wang, and J. Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 149–158, Washington, DC, USA, 2009. IEEE Computer Society.
4. L.K.J. Grace, V. Maheswari, and D. Nagamalai. Web log data analysis and mining. In Natarajan Meghanathan, BrajeshKumar Kaushik, and Dhinaharan Nagamalai, editors, *Advanced Computing*, volume 133 of *Communications in Computer and Information Science*, pages 459–469. Springer Berlin Heidelberg, 2011.
5. A Makanju, A.N. Zincir-Heywood, and E.E. Milios. Investigating event log analysis with minimum apriori information. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 962–968, May 2013.
6. M.G. Schultz, E. Eskin, E. Zadok, and S.J. Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49, 2001.
7. M. A. Siddiqui. *Data mining methods for malware detection.* ProQuest, 2011.
8. R. Winding, T. Wright, and M. Chapple. System Anomaly Detection: Mining Firewall Logs. In *Securecomm and Workshops, 2006*, pages 1–5, Aug 2006.