Host Based Intrusion Detection System for Log Files

Abstract-- **World over the internet has become less secure and computer security is the major issue today. Many possible solutions are available today and intrusion detection system is one of them to handle this issue. Intrusion Detection systems uses pattern matching algorithm as the core algorithm for its functioning. Intrusion Detection pattern matching technique had applied on the Security Event log files for the Windows/Linux system. Matching of the intrusion pattern had indicated the intrusion in the system. Existing signature based IDS have noteworthy overheads in terms of execution time due to the pattern matching operation. This project aims to speed up the pattern matching operation through parallelizing a matching algorithm on a multi-core CPU. In this paper, Myer algorithm had paralleled on a multicore CPU under the MapReduce framework. On average, we attained four times speedup using our multi-core implementation as compared other serial algorithms. The system is hoping to grow into HIDS that will capture all kind of intrusions in future.**

Keywords- Host based intrusion detection system; pattern matching; Linux/Windows; System logs; MapReduce.

I. INTRODUCTION

Computer security is becoming an area of utmost importance due to the rapid increase number of people who use computers. As more and more computer systems are setup, there are certainly bugs in these systems which attackers attempt to exploit. Host Intrusion detection systems (HIDS) are designed to monitor a computer internals to detect and prevent against malicious activity. HIDS can monitor users, applications, networks or combination of the all above from well-known and unknown attacks by analyzing its security log files and network interface for incoming packets.

The two main methodologies to host intrusion detections are misuse detection and anomaly detection.

Misuse detection attempts to model well-known attacks. Then any behavior or pattern that matches the model is identified as an attack. It is a Signature-based detection method where the predefined rules or user defined rules are used to identify the intrusion. The rules will determine pattern of the security logs that need to be detected. If the security logs patterns matches with the defined pattern, the intrusion has occurred. The collection of these signatures composes a knowledge database that use by the HIDS to compare all log options that pass by and check if they match a known pattern [1]. Misuse detection techniques in general is ineffective against novel attacks that have no matched rules or patterns yet [2]. The advantage of this approach is well-known type of attacks can be identified by this matching technique. The drawback is, it is difficult to detect a new attack pattern or a modified attack which may by pass the system.

Second approach is Anomaly based detection method where the system will learn the normal and anomaly packet traffic and it detects intrusion on modified attack or unknown attack. This approach requires some artificial intelligence's element where the system can learn the normal pattern of the packet traffic on the interface and make detection on intrusion if the behavior of the packet is changed. The advantage of this approach is, it can be used to detect unknown attacks. The drawback of this approach is, it is slow in detecting intrusion. The intrusion may have occurred number of times or continue to occur after the IDS detect the intrusion. The second drawback of this model is that network has produced all types of behavior in learning phase of the HIDS that hides from the user, so it may cause a high number of false-positive alerts .

The two different approaches to intrusion detection are Host-based intrusion detection (HIDS), and Network based intrusion detection (NIDS). HIDS attempts to detect against attacks on a particular machine system. This is typically done through analysis of a computer's internal. Network-based intrusion detection attempts to detect against attacks on a network. This is typically done through analysis of network traffic. There are few types of IDS such as Network IDS, Host-based IDS, Protocol-based IDS, and Application Protocol-based IDS. This study is focused in development of Host-based IDS. Host-based IDS is a type of IDS that is allocated on a particular host on the network. Its major benefit is the detection of intrusion to intended host or local host. Host- based IDS provide an extra protection to the host where it monitor more aspect of host such as monitoring file system integrity, host access, network packet that send to the host, system registry and system security log files. where log file of host system are analyzed from time to time to detect intrusion. The content of the log file is compared to IDS rules or pattern that is predefined in database created. File system monitor can check files on a large number of different characteristics such as permissions, anode, and number of links, owner/group, size, directory size, checksum, type, link, and active changing [3].

II. Background

This section provides the necessary background to under-stand the problem in hand. Sections 2.1 introduces Myers algorithm and dynamic programming. Section 2.2 briefly reviews the related parallel programming techniques, and Section 2.3 presents the MapReduce framework and the two different implementations.

2.1 Myers algorithm

The Myers algorithm is an approximate string matching algorithm. An approximate matching algorithm matches a large text of length *n* with a short pattern *p* of length *m* allowing up to *k* differences, where *k* is a chosen threshold error . The Myers algorithm relies on a simple dynamic programming (DP) concept . It uses recursive formulas and simple bit operations to compute the edit distance between the text and patterns to find the equalities or differences [4]. The edit distance between two strings isexpressed as the minimum edit operations required to transform a text *t1* to another text *t2* or vice versa. Commonly, there are three typical variations of edit distance. The first form is called the Hamming distance [5]. It computes the number of positions in the text that has different characters, i.e., how many characters are needed to convert a text *t1* to another text *t2*. The compared texts or strings must be of the same length. The second form is called the Levenshtein distance, which does not have any restriction over the text size [6]. The edit distance is the minimum number of edit operations: insertion, deletion, and substitution, which are needed to convert two strings into each other. The third one is the Damerau edit distance. It allows the transposition of two adjacent characters to complete the conversion between the two strings [7]. Figure 2 shows examples of the edit operations.

The Myers algorithm uses the Levenshtein distance to compute the matches. It considers two strings similar if the edit distance (*ed*) between the two strings (A, B) is less than or equal to a predefined threshold (*k*) (*ed* (*A*,*B*)<=*k*).

The formal approach to solve the problem of approximate string matching and to find the minimum edit distance is to use dynamic programming. Dynamic programming is an approach which uses a recursive formula to compute new values based on a prior knowledge of previous values. For two strings of lengths *m* and *n*, a matrix of size *m* × *n* is filled column-wise or row-wise, respectively, with distances between the strings. The patterns are arranged vertically and the logs are arranged horizontally. Initially, the matrix cells are initialized as follows. Cells *C*[0,*j*] = 0 and cells *C*[*i*,0] = *i*, and next, the matrix is expanded according to the recursive formulas in Eq. Given below [8].

Each cell of the matrix, *C*[*i*,*j*], represents the edit distance between the two strings *P[1 →i], T[1 →j]* ending at positions *i*,*j*. The addition of one in Eq. 1 is the penalty if a match does not exist. This case represents a substitution.

2.2 Parallel programming

Parallel programming is a technique in which many computations are performed concurrently. Parallel computation divides a big task into smaller sub-tasks to be executed simultaneously. Parallelism can utilize multi-core processors in a single machine or multi-processors in a cluster of machines.

The parallel execution on a multi-core or a cluster can take many forms. It can be categorized into bit-parallelism, data parallelism, or task/function parallelism. The focus of bit-parallelism is to minimize the count of instructions to execute an operation. This can be done by increasing the processor word size. In data parallelism, the data is split into many pieces and is distributed to multiple cores or processes. All processors run the same code simultaneously but on different data piece. This is also known as single instruction multiple data (SIMD) approach. In contrast to data parallelism, task parallelism is a form of parallelism where multi-processors run different codes or tasks on the same piece of data simultaneously.

2.3 MapReduce

MapReduce is a programming model released by Google to handle the processing of large dataset in parallel [17]. The idea behind this framework is to hide the complexity of parallelism from the programmer. Moreover, the framework provides the programmer with a simple API to present the logical perspective of an application.

Recently, MapReduce has been widely used in both academia and industry. MapReduce has become a standard computing platform used by large companies such as

Google, Yahoo!, Facebook, and Amazon. Statistics show that Google uses MapReduce framework to process more than 20 petabytes of data per day [17].

2.3.1 MapReduce basic programming model

MapReduce framework consists of two primitive functions defined by the user: Map and Reduce. Additionally, it has a runtime library to automatically manage the parallel computations without the need for user interventions. The library handles data parallelization, fault tolerance, and load balancing.

In MapReduce model, the input and output data take the form of key/value pairs <*key*,*value*>. Map and Reduce are the main two operations that are applied to the key/value pairs. A large input data is split into chunks of a specified size. For example, Google's implementation partition the data into *M* pieces each of size 16–64 MB. The Map function takes the input as a series of key/value pairs <*k*1,*v*1> and performs the task assigned by the programmer. The output of the Map function is a series of intermediate key/value pairs <*k*2,*v*2>.

The framework performs a shuffle phase in order to group the values of the same key. Afterwards, the intermediate data pairs are sent to the appropriate Reduce function. The Reduce phase takes the combined intermediate values as a key and a list of values and then executes the user-defined Reduce function to produce the final result.

2.3.2 Phoenix MapReduce

Phoenix is a MapReduce implementation introduced by Stanford University [9]. It targets multi-core and multi-processor systems. Phoenix relies on the same principles of the original MapReduce implementation. It provides a runtime system library and a set of APIs that handle the underlying parallelism issues automatically. Unlike Google's MapReduce, Phoenix uses threads instead of clusters to perform the Map and Reduce tasks. Additionally, it uses shared memory for the purposes of communication.

Phoenix provides two distinct types of APIs: the first set is defined by the user such as Map, Reduce, and Partition. The second set includes the runtime APIs that deal with system initialization and emitting of the intermediate and final output as key/value pairs. Pthreads library is the basic development package of Phoenix runtime. The execution flow of the runtime passes through four basic stages: Split, Map, Reduce, and Merge. At the beginning, the user program initiates a scheduler that controls the creation of Map and Reduce threads. A buffer is used to provide the communication between workers. Data and function arguments, such as the number of workers, the input size, and the function pointers, are passed to the schedule.The Map phase splits the input pairs into equal chunks and provides a pointer for each data chunk to be processed by the mappers. The intermediate data that is generated by the mapper is partitioned into units as well and ordered by the partition function to be ready for the reduce phase. The reduce phase does not start until the entire Map task is finished.

The scheduler assigns the tasks to the reducer dynamically. Each unique key with its associated list of values is processed at a reducer node or thread. At the end, the final outputs of the tasks are merged into a buffer and sent back to user program.

2.3.3 MAPCG MapReduce

Nowadays, hardware accelerators are being widely used to perform general-purpose computations. Some researchers have implemented MapReduce on hardware accelerators such as FPGAs and GPUs [10]. These implementations try to exploit the parallelism capabilities of the accelerators to improve the execution time of the MapReduce framework. GPUs have received a great attention due to their high performance capabilities.

MAPCG framework relies on the accelerators context that says "Write once, run anywhere". It was designed to provide a portable source code between CPU and GPU using MapReduce framework [11]. MAPCG's runtime library allows the user to focus on the logical perspective of the algorithm implementation rather than dealing with the side-burdens of parallelism such as load balancing and communication issues. In short, MAPCG was designed to parallelize data-intensive applications on multi-core CPUs and GPUs. It automatically generates a portable code that can schedule the MapReduce tasks on both CPU and GPU. In addition, it implements a dynamic memory allocator for the CPU and GPU that behaves efficiently even when using a massive number of threads. Moreover, a hash table was designed to group the intermediate data on GPUs to enhance the sorting phase of MapReduce keys.

Basically, MAPCG has two main parts, a high-level programming language and the runtime library. The high-level language facilitates and unifies the programming task, while the runtime is responsible of executing the code on different processing units such as the CPU or GPU. As other MapReduce frameworks, the execution begins by splitting the data inputs into chunks and sending them to the Map phase. The function Map is applied to each data chunk and outputs the intermediate data. The Reduce function performs the appropriate computations on the intermediate data to produce the final outputs.

III. IMPLEMENTATION

The focus for this HIDS is to make it detect intrusion through security log file that provided by operating system. Therefore, this section will discuss about the implementation and result for this study. Actually, it is about its processes. The Processing Units perform the task of capturing the newly logged event which is the initiating step of the entire process. The signatures are created and stored in database. After the generation of signatures logs files were analyzed by using Myer's pattern matching technique.

3.1 Creating User Signatures

For well-known attacks User signatures are already present and can be stored in database, but for some other attacks signatures are not available. So for such attacks, User signatures are created from the log files generated by the Basic Security Module (BSM) of the operating system. The BSM is a kernel module that logs all events that occur on a given machine. Each of these events is actually a system call, and for each system call a record is generated, consisting of tokens corresponding to the type of event. All the system calls that are generated by the user are extracted from log files and then commands are extracted from it. For doing this the UNIX utility commands are used, auditreduce and praudit. The auditreduce command is used for audit management and record selection, while praudit is used to convert binary log files in to ASCII format. At the command prompt, this is done as follows:

“auditreduce -u [username] [log file in binary form] | praudit – l”.

The auditreduce command takes the log file specified and extracts all system calls generated from the given username. The output of this is piped to the praudit command, which will then write each record (system call) to stdoutput one record per line.

3.2 Pattern matching

In order to find the suspicious logs, most of Host IDSs employ a pattern matching algorithm. The algorithm checks the presence of a signature in the security logs and outputs the location of the security log. The algorithm must be fast enough to detect the malicious behavior, and it must be scalable in order to meet the increase in both the number of signatures and the link speed.

String matching algorithms can be categorized into single and multiple pattern matching algorithms. In the single pattern matching, one pattern is matched against the entire text at a time. In contrast, the multiple pattern matching approach compares the text sequence against all signatures all at once [12]. Obviously, the multiple matching approach is a better choice for intrusion detection to avoid sweeping the packet many times. However, it consumes more memory and requires a pre-processing phase to program the patterns before matching can commence. The Myers algorithm is a better approach for this.

3.3 Logging and Monitoring

Logging and Monitoring gives corresponding responses for verified intrusion behaviors, including static measures, for example, record attack data, store captured data, send emails and messages to the manager, it can also take some active dynamic preventive measures, cut off the intruded connection and modify the access control of router, for instance. It mainly includes patrol agent and host mobile agent two types. The former is to carry out host behavior detection and response, and the latter is mainly to respond to the invasion information in the intrusion detection module and report the invasion logs to the control server, as well as store in the database. The system is only to achieve log storage, and the database storage will be achieved until the system expansion and improvement. Generally, the system mainly includes three very important functions , namely Alert () function is to give real-time warning of intrusion behaviors, logtoTable() is to set up a log of high danger class intrusion behaviors in database and Trace Intrusion is to trace and take evidence of intrusion behaviors if it proves necessary.

IV. CONCLUSION

This paper is developed about a Host based intrusion detection system for security log files, which is able to detect malicious activity and show an alert of intrusion to user through the system. The usage of the log files analyzer HIDS is limited used to the host-based level. The study had done by the system can read security event log files from Windows/Linux. Then, the system can make comparison for that file with the intrusion pattern files that reside inside the database. Therefore, if these two files had matching structure, the system will assume this file is an intrusion and give an alert to the user.

Security event log files from Windows/Linux are needed to be analyzed by any system like log file analyzer HIDS for security purposes in host or Computer. The types or structure of intrusion pattern needs to be revised and updated regularly. The most important things is that the organizations have a system to detect many threats in a computer host. It will support to prevent an organization from most of the threats [13].

REFERENCES

[1] Pedro Bueno (2002), Intrusion Detection Systems, Linux Journal, Volume 2002, Issue 97 (May 2002), Specialized Systems Consultants, Inc.

[2] Wenke Lee et al., (2000), A Framework for Constructing Features and Models for Intrusion Detection Systems, ACM Transactions on Information and System Security (TISSEC), Volume 3 Issue 4, ACM Press.

[3] Boer P. et al., (2005), Host-based Intrusion Detection Systems, Revision1.10 – February 4, 2005, SNB student projects 2004 - 2005M. Young, the Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[4] Edit_distance (2017). http://en.wikipedia.org/wiki/Edit_distance.

[5] RW Hamming, Error detecting and error correcting codes. Bell Syst. Tech. J. **29:**

[6] V Levenshtein, Binary codes capable of correcting deletions. Insertions Reversals. Sov. Phys. Dokl. **10**(8), 707 (1966).

[7] F Damerau, A technique for computer detection and correction of spelling errors. Commun. ACM. **7**(3), 171–176 (1964).

[8] S Wandelt, D Deng, S Gerdjikov, S Mishra, P Mitankin, M Patil, E Siragusa, A Tiskin, W Wang, J Wang, U Leser, *State-of-the-art in string similarity search and join*, vol. 43, (2014).

[9] C Ranger, R Raghuraman, A Penmetsa, G Bradski, C Kozyrakis, in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. Evaluating MapReduce for multi-core and multiprocessor systems (Scottsdale, 2007), pp. 13–24.

[10] Y Shan, B Wang, J Yan, Y Wang, N Xu, H Yang, in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '10)*. FPMR: MapReduce framework on FPGA (ACMNew York, 2010), pp. 93–102.

[11] C Hong, D Chen, W Chen, W Zheng, H Lin, in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques (PACT '10)*. MapCG: writing parallel program portable between CPU and GPU (ACMNew York, 2010), pp. 217–226.

[12] S Wu, U Manber, A fast algorithm for multi-pattern searching, Technical Report TR-94–17Department of Computer Science, University of Arizona, 1994).

[13] Firkhan Ali, H. A., "An Analysis of Possible Exploits in the Computer Network's Security "in ISC 2005: Proceedings of the International Science Congress 2005. PWTC, Kuala Lumpur, 2005. pp.338.