TotalADS: Automated Software Anomaly Detection System

Syed Shariyar Murtaza[1], Abdelwahab Hamou-Lhadj[1], Wael Khreich[1], Mario Couture[2]
[1]Software Behaviour Analysis Research Lab, ECE, Concordia University, Montreal, QC, Canada
{smurtaza, wkhreich, abdelw@ece.concordia.ca}
[2]Systems Protection and Countermeasures, Defence Research and Development Canada, Valcartier, Canada
mario.couture@drdc-rddc.gc.ca

ABSTRACT
When a software system starts behaving abnormally during normal operations, system administrators resort to the use of logs, execution traces, and system scanners (e.g., anti-malwares, intrusion detectors, etc.) to diagnose the cause of the anomaly. However, the unpredictable context in which the system runs and daily emergence of new software threats makes it extremely challenging to diagnose anomalies using current tools. Host-based anomaly detection techniques can facilitate the diagnosis of unknown anomalies but there is no common platform with the implementation of such techniques. In this paper, we propose an automated anomaly detection framework (TotalADS) that automatically trains different anomaly detection techniques on a normal trace stream from a software system, raise anomalous alarms on suspicious behaviour in streams of trace data, and uses visualization to facilitate the analysis of the cause of the anomalies. TotalADS is an extensible Eclipse-based open source framework that employs a common trace format to use different types of traces, a common interface to adapt to a variety of anomaly detection techniques (e.g., HMM, sequence matching, etc.). Our case study on a modern Linux server shows that TotalADS automatically identifies contemporary attacks on the server, shows anomalous paths in system traces, and provides forensic insights.

Keywords
Anomaly Detection, Trace Analysis, Software Security.

1. INTRODUCTION
Despite recent advances in software technologies, the security of software systems continues to remain at risk. Everyday new vulnerabilities are discovered and their exploits continue to threaten software systems. To protect software intrusions, anomaly detection systems (ADSs) are designed to detect anomalies by measuring deviations from a normal baseline of a host that is built in a lab (attack-free) environment using system traces. Unlike signature-based intrusion detection techniques, which use signature of known malwares, ADSs can detect unknown emerging attacks.

In recent years, there has been a significant increase in the number of anomaly detection techniques, which can be used to detect anomalies on a host with a lower, acceptable false alarm rate, and a high anomaly detection rate (e.g., see KSM [15], Semantic ELM [4], and Hidden Markov Models [8]). Host-based anomaly detection systems can benefit military systems, telecommunication routers, and other safety critical systems whose baselines of normal behaviour do not change rapidly. If host-based anomaly intrusion detection systems are used alongside signature-based intrusion detection methods, then many known and unknown anomalies in a system could be easily and immediately detected. However, host-based anomaly detection techniques are not publicly available; developers will have to implement these techniques themselves to use them for anomaly detection. Also different developers/administrators may collect different types of traces and software logs to diagnose anomalies. To our knowledge, there is no tool or framework that allows detecting anomalies using different techniques on different types of traces and facilitates visual forensic analysis of anomalies.

In this paper, we overcome this challenge by proposing a novel framework for automated host-based anomaly detection, called TotalADS. TotalADS is an open source tool developed as a plug-in for Eclipse. It integrates different anomaly detection techniques, different trace readers and a rich set of trace views in one common platform. It provides an interface to add new anomaly detection techniques, the ability to add new readers for different types of trace formats, and the capability to add new views to visualize trace contents. TotalADS can also build and evaluate models from anomaly detection techniques in real-time on live trace streams. Once an anomalous trace is detected, it can then be analyzed in detail using TotalADS views, such as, the process flow view, the CPU utilization view, etc.

In short, our contributions are as follows:

- A novel open source framework to integrate anomaly detection techniques and different trace formats, e.g., Common Trace Format [3] , XML and text format.
- Implementation of three host-based anomaly detection techniques: Sequence Matching [7] [25], Hidden Markov Model [27] [24] [8], and Kernel State Modeling (KSM) [15], described later in the paper.
- A novel framework that allows training and testing of anomaly detection techniques on trace streams in real-time.

The remainder of the paper is organized as follows. Section 2 explains the background and related techniques. Section 3 describes TotalADS in detail. Section 4 shows a case study on a Linux server by using TotalADS and Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK
2.1 Host-based ADS
Anomaly detection systems can be classified into Host-based Intrusion Detection Systems (HIDS) or Network-based Intrusion Detection Systems (NIDS). NIDS examine network traffic to detect anomalies; e.g., the use of Bayesian network on network traffic records to detect anomalies [23]. HIDS focus on using metrics present on a host to detect anomalies. A type of HIDS uses

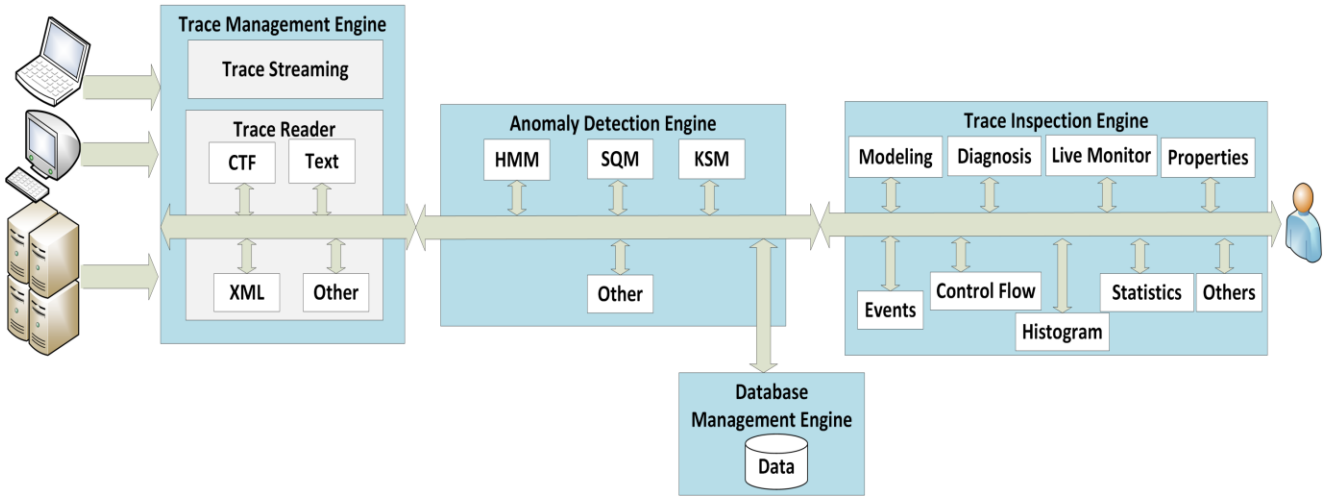

Figure 1: Overview of TotalADS Architecture

different techniques on normal audit records (logs) of a host (e.g., CPU usage, process id, user id, etc.). These systems measure an anomaly threshold and raise alerts when particular attribute values of a new record are above the threshold [27]. Another type of HIDS train different techniques on system call traces of normal software behaviour. These systems raise alerts when the deviation from normal system calls is observed in unknown software behaviour (e.g., a trace). Anomaly based HIDS focusing on system calls' deviations are related to our work and are described below.

Some well-known works in the area of system calls based anomaly detection include the use of sequence matching [6] [7], Hidden Markov models [25] [24] [27], and neural networks [9][2]. The sequence matching and Hidden Markov model techniques work by building a model of a system from system call traces during normal operations (e.g., in labs) and then detecting anomalous sequences in systems operating in fields. Neural networks based techniques mostly take both normal and anomalous system call traces to build models of the system that can characterize normal and anomalous behaviour. In our earlier work, we transformed system calls into states of kernel modules, built a model of the system from benign kernel modules based traces, and detected anomalies in unknown traces [15]. In this paper, we propose a tool (or an extensible framework) in which different anomaly detection techniques can be integrated easily.

2.2 Contemporary Tools

Contemporary tools can be divided into two main categories: (a) tools for machine learning; and (b) tools for trace inspection. Tools for machine learning incorporate variety of machine learning techniques for experiments on different type of datasets; e.g., Weka [26], R [21] and MOA [12]. These tools are mostly suitable for experiments on any kind of data; however, data have to be transformed into their format. These tools also do not support host-based anomaly detection techniques like Sequence Matching, Kernel State Modeling. HMM has to be programmed too, to be used as a host-based anomaly detection technique. TotalADS focuses specifically on analysis of software traces and logs to build host-based anomaly detection models. It can also be used to collect traces and build models in real time. TotalADS also provides the necessary views to inspect traces to facilitate in forensic analysis which do not exist in machine learning tools.

The tools for trace inspection only provide contextual views or textual summary to investigate traces. For example, Babletrace can be used to convert binary LTTng traces in Linux into a text format [10], and later the grep utility can be used to find patterns in it. Strace analyzer can provide statistics about a trace in a textual format [20]. Similarly, tools for user space trace analysis, such as Bugsense [1] and Cittercism [2] provide only summaries in the form of charts, trends of a trace. None of these tools automatically point out anomalous traces and the origin of anomalies.

In addition to trace viewers, the host-based anomaly detection tools like OSSEC [17], Samhain [19], etc. are also related to TotalADS. These tools mostly use predefined rules to monitor logs, opened ports, integrity of running kernel, etc. In TotalADS, there are no predefined rules. TotalADS uses machine learning techniques to capture a normal baseline of a system and detect anomalies in real-time. For example, the port scanner cannot detect an attack that bypasses the authentication mechanism on an already opened port by MySQL; however, an anomaly detection technique can detect such attack (see case study, Section 4).

3. TotalADS

TotalADS provides a framework, based on Eclipse IDE, for integrating multiple anomaly detection techniques in a way that they can be evaluated on different types of execution traces. TotalADS is based on Java and its architecture is shown in Figure 1. It consists of four main components: Anomaly Detection Engine, Trace Management Engine, Trace Inspection Engine, and a Database Management Engine. In the following sections, we explain each of these components in detail.

3.1 Trace Management Engine

Trace Management Engine allows to read different trace formats. It has a CTF (binary) trace format reader for the LTTng tracer [10], a text file reader using regular expressions and a XML log reader. The Common Trace Format (CTF) for traces is a format standardized by the Multicore Association in collaboration with the Linux community [3]. CTF is actually a compressed format to represent a large number of events with their timestamps. Any trace, once converted to a CTF format, can be directly used with TotalADS without adding any additional readers.

In Linux-based systems, the LTTng tracer can be used to collect both kernel (system call traces) and user space (function call) traces [10]. LTTng is a lightweight Linux tracer that stores traces in a Common Trace Format (CTF) [3]. LTTng also provides a utility to transform any trace into CTF format and TotalADS can easily read traces in the CTF format. TotalADS can also read traces from other tracers, such as STrace which generates a text format of traces [20]. Any kind of text file can be parsed in

TotalADS using the regular expression wizard. Similarly, XML logs can be parsed automatically by using wizards in TotalADS. Trace Management Engine also provides an extensible common interface (in Java) to add new trace readers.

In addition to reading existing traces in the system, the Trace Management Engine allows to collect LTTng (kernel) traces from a remote system via Secure Shell (SSH) protocol. It then extracts the desired events and passes them to the Anomaly Detection Engine. The live trace streams can then be used by the Anomaly Detection Engine to evaluate different techniques in real-time.

3.2 Anomaly Detection Engine

This is the most important component of TotalADS. It integrates different host-based anomaly detection techniques by using a common interface in Java. Any anomaly detection technique can be used with TotalADS to build models from different types of traces either offline or through live streaming. A model is the representation of traces that a technique develops during training. A technique can be used to build many models. Once a model is built, it can be validated on another (or same) set of normal traces. During validation, the technique may adjust decision thresholds of a model to lower its false alarm rate before putting the model into testing (production). During the testing phase, a model is used to evaluate traces coming from a system in operation to detect anomalies in incoming traces. Decision thresholds can also be adjusted during testing by a user if suspicious alarm rate by a model is required to be decreased or increased. The best model is the one which has a low false alarm rate and a high true alarm rate on test traces. TotalADS integrate three different anomaly detection techniques, namely, Kernel State Modeling (KSM), Sequence Matching (SQM) [7] [25] [6], and Hidden Markov Model (HMM) [27] [24] [8].

KSM focuses on detection of anomalies by transforming system calls into states of kernel modules [15]. A system call can be classified into eight different states: file system state, kernel state, memory management state, networking state, inter process communication state, security state, architecture state and (a rare) unknown state. KSM then identifies anomalies by comparing the probabilities of occurrences of states in normal and anomalous traces [15].

SQM works by extracting sequences of length 'n' from a trace by sliding a window one event (e.g., system call) at a time [6] [7] [25]. For example, for a trace having system calls "3, 6, 195, 195", two sequences "3, 6, 195" and "6, 195, 195" of length 3 can be extracted. SQM extracts sequences from normal traces and then compares them against the sequences in an unknown trace. If a new sequence is found in an unknown trace then it considers it as anomalous. The Hamming distance between sequences can be used to adjust the decision threshold to reduce false alarms; e.g., a sequence "3, 5, 195" is anomalous for above sequences as a mismatch occurs only at one position—i.e., a Hamming distance of only one. If the minimum Hamming distance matching criterion is set to more than one, then it is a normal sequence.

HMM is a stochastic model for sequential data and hence it is naturally suitable for modeling temporal order of system call sequences [18]. The process is determined by a latent Markov chain having a finite number of states, N, and a set of output observation probability distributions, B, associated with each state. Starting from an initial state N_0, the process transits from one state to another according to the matrix of transition probability distribution, A, and then emits an observation symbol Ok from a finite alphabet (i.e., M distinct observable events) according to the output probability distribution, Bj(Ok), of the current state Nj. HMM is typically parameterized by the initial state distribution probabilities (Π), output (emission) probabilities (B), and state transition probabilities (A). Baulm-Welch technique is used to train the model parameters to fit the sequences of observations, T, [18]. During the validation phase, HMM adjusts the decision threshold (log likelihood) of prediction of anomalous alarms on T sequences from traces. In the testing phase, if the probability value of any sequence in a trace is below the selected threshold, then we consider the trace as anomalous otherwise we consider it as normal. We use Apache Mahout, the scalable machine learning library to implement HMM [11].

3.3 Trace Inspection Engine

Trace Inspection Engine encompasses a set of views that can be used to build anomaly detection models, diagnose traces using models, train and evaluate models on live traces, inspect individual event details in a trace, understand control flow of processes, comprehend utilization of CPUs in a trace at different time, get event statistics in a trace, and visualize occurrences of events as histogram of frequencies over a time line. Trace Inspection Engine is based on TMF (Tracing and Monitoring Framework) [22], a tracing platform developed by Ericsson. TMF views can be extended too by using a common Java interface. Due to the lack of space, we have omitted the details of different views in the trace inspection engine but some of the views are shown in the case study in Section 4.

3.4 Database Management Engine

Database Management Engine provides an interface to store data in a NoSQL based database management systems that stores data in the form of JSON (JavaScript Object Notation). The advantage of JSON is that models can be serialized directly from their class representation in memory to a database without any conversion into a relational schema like structure. Currently, we are using MongoDB to store models and corresponding information [14].

4. Case Study: Anomaly Detection on a Linux Server

This section discusses the experiments performed on a Linux server using TotalADS. In particular, we address the following two research questions:

(RQ1) Can TotalADS automatically train and test models on live trace streams?

(RQ2) Can TotalADS automatically diagnose anomaly and facilitate in diagnosis of the origin of anomaly?

4.1 Dataset

We used an Ubuntu Linux server as a case study. We equipped the server with Ubuntu Linux 12.04 operating system with Apache 2.2.17 web server, PHP 5.3.5 server side scripting engine, TikiWiki 8.1 content management system, FTP server, MySQL distribution 5.1.56 (version 14.14) database management system and an SSH server. These versions of the application software systems were selected because of the well-known vulnerabilities. MySQL 14.14 has a vulnerability which allows remote attackers to bypass the authentication mechanism by repeatedly authenticating the same incorrect password (CVE 2012-2122)[1]. TikiWiki 8.1 has a vulnerability allowing remote attacker to execute any arbitrary PHP code (CVE 2012-0911). PHP 5.3.5

[1] CVE represents a Common Vulnerabilities and Exposures (CVE) ID is a unique identifier assigned to vulnerabilities [16].

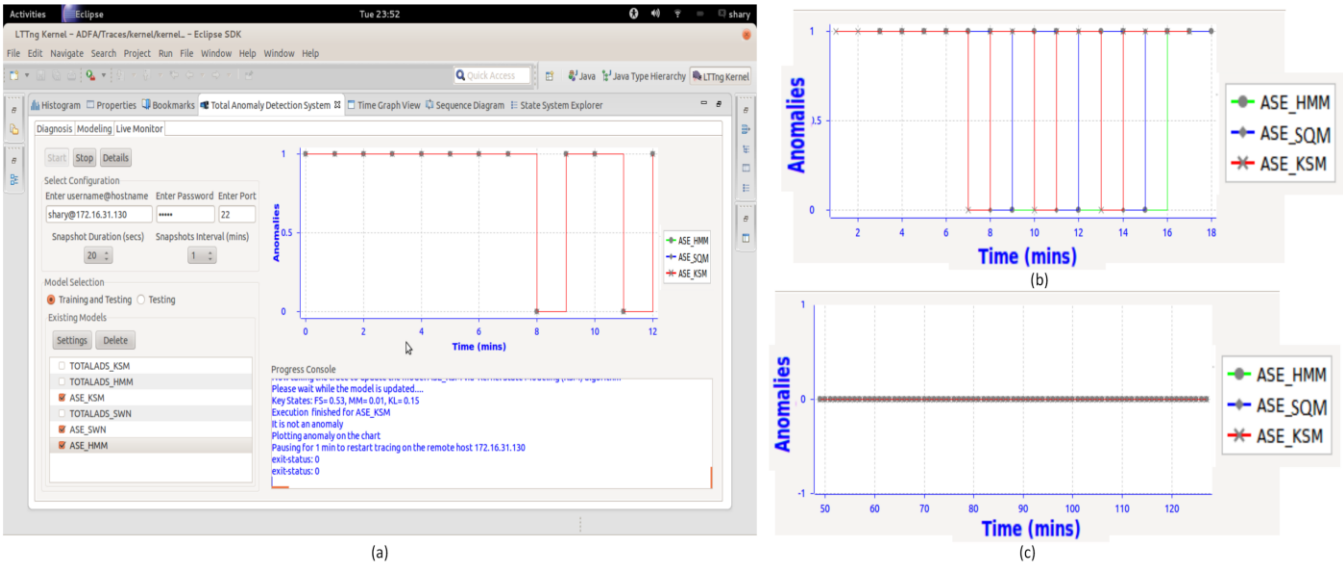

Figure 2 Live Training and Testing in TotalADS

vulnerability allows the remote attacker to get context sensitive information from the process's memory (CVE 2011-1153). Apache web server has a vulnerability to allow remote attackers to cause a daemon crash via an empty cookie (CVE 2012-0021). Similar configuration was also used in a dataset publicly available on the site of University of New South Wales [5]. To exploit these vulnerabilities, we used an open source penetration testing tool Metasploit Framework (MSF) [13]. We also used MSF to generate brute-force attacks on FTP and SSH servers. We collected anomalous data by executing attacks using MSF.

In order to obtain data of normal activities on the Linux server, we performed several tasks that reflect the normal utilization of the system. These tasks include MySQL authentication and manipulation on Tiki database, document processing in Open Office Writer and Calculator, directory and files manipulation, installation of additional software such as Vim, web browsing, web page serving, and MySQL, FTP and SSH authentication and execution of queries from a remote system.

4.2 The Process

We started the process of trace collection by employing the Live Monitoring view of TotalADS. The process is divided into the two modes: Live Training and Testing, and Live Testing only.

In the Live Training and Testing mode, we first initialized a model of KSM, a model of SQM, and a model of HMM in TotalADS. Each technique takes different initialization parameters. We selected the kernel version of Ubuntu 12.04 for KSM, and window length of 6 for SQM. For HMM, we selected 10 states, 200 observable events, and 10 iterations of the Baum-Welch technique. Second, we performed live training and testing over SSH on our Linux server. The idea is to only use the system normally without any attacks. We built the normal models of the system using live streaming. During this live training and testing, TotalADS first tests a new trace against an existing model in a system, shows its results, and the trains the model on the same trace. We performed different normal activities on the Linux server, as described in Section 4.1, during this phase.

In the second mode, live testing, we started trace collection on the Linux server by only selecting the live testing option in TotalADS. We used the server normally, started Metasploit [13] on another system and attacked the Linux server. The idea is to simulate both normal and abnormal behaviour of the system. We used different attacks one by one to exploit vulnerabilities in the applications running on the Linux server. If an attack is detected by a technique, the trace of the attack is further investigated in the trace analysis views.

4.3 Results

4.3.1 Automatic Training and Testing on Live Trace Streams

The results for Live Training and Testing phase are shown in Figure 2. TotalADs started by testing the new traces on the models of the KSM technique, the HMM technique, and the SQM technique. If a trace is found as anomalous then TotalADS plots it as anomaly (one) on a chart. It then trains three models on the new trace. Figure 2a shows the complete view of TotalADS; whereas, Figure 2b and Figure 2c only show the zoomed-in view of the chart in Live training and testing mode.

Initially, each of the three models of the three different techniques predicted every incoming trace as an anomaly. TotalADS at this point shows higher false alarm rate because the models built for techniques did not have sufficient (or rather zero) information to predict anomalies (Figure 2a). The three series in the chart represent three different models. When there is only one line visible on the chart then it means that three models predict the same results. All initial traces are shown as anomalies on the chart in Figure 2a, except at 8th and 11th minute.

When time passed, regular binary signals started to appear on the chart, showing that several newer traces have started to be detected as normal traces (Figure 2b). At this point, we also slightly increased the decision thresholds of the models. In machine learning, decision thresholds are usually adjusted during a validation of a model before testing. The decision threshold for SQM, HMM, and KSM were Hamming distance, alpha, and probability values, respectively. This process of training kept on continuing until we reached a point in time where all the new upcoming traces started to appear as normal. This is shown in Figure 2c. We stopped live training and testing mode at this point.

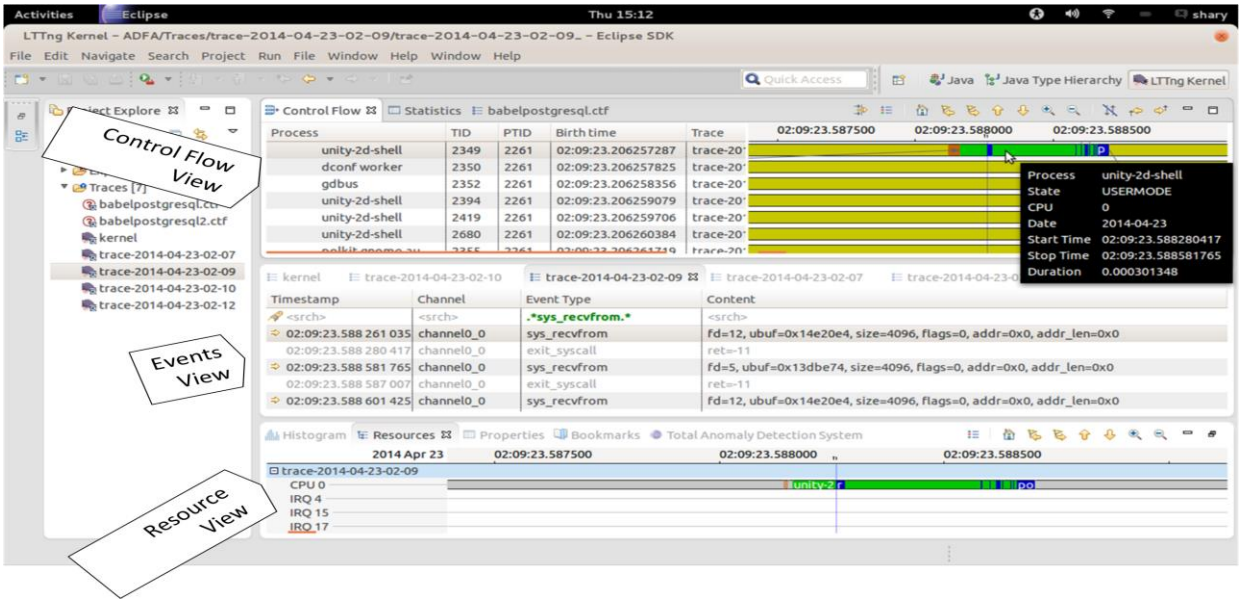

Figure 3: Inspection of the attack to bypass authentication mechanism in MySQL

These results show that TotalADS can be used to train and test anomaly detection techniques on live traces (streams of traces) from a real system with minimal human intervention. This answers the first research question (RQ1).

4.3.2 Automatically Detect Anomaly and Inspect the Origin of Anomaly

In the Live Testing mode, we used the server in a normal manner and also exploited its vulnerabilities using Metasploit [13]. The results are shown in Figure 4. In the case of Figure 4, the attack executed was an attempt to illegally bypass MySQL authentication mechanism with a wrong password. Also note that the normal operations include the legal remote authentication on MySQL. Initially, see Figure 4, when the server was used normally the predictions of anomalous alarms remain zero from all three models but as the remote hacker (Metasploit) tried to penetrate the system, the models raised anomalies in a span of 3 (between 9 and 11) minutes. The models did not raise anomalies on exactly the same traces but the anomalies were raised on the traces collected in a close proximity.

For example, the SQM technique showed 1 out of 10 traces as anomalous and the anomalous sequence in the trace contained 4 (66%) occurrences of the "sys_recvfrom" system call in a sequence of total of 6 system calls. The anomalous sequence was *"sys_rt_sigtimedwait, sys_recvfrom, sys_recvfrom, sys_recvrfrom, sys_poll, sysrecvfrom"*. The "sys_recvfrom" system call is used to receive a message from a socket. This seems suspicious enough to explore it further. We therefore loaded this trace into the trace inspection views. The trace inspection views are shown in Figure 3. The three trace inspection views shown in Figure 3are Control Flow view, Events view, and Resource View.

In the Events view, we searched for the event "recvfrom" and found all the occurrences of "sys_recvfrom" emphasized more than other events by the Events view. The detailed attributes, in this case arguments of the system call, are also shown in the Events view. When we clicked on that "sys_recvfrom" at the time stamp "02:09:23.588.581 765", the Control Flow view got synchronized and immediately showed the process corresponding to the timestamp and the event. The yellowish green bar represents the wait-blocked state of the process, brown bar represents the CPU-wait, the green bar represents the user mode, and the blue bar represents the system call mode of a process. When a mouse is hovered over the bar, the details are also show as a tool tip (see Figure 3). In this case, this is a process "unity-2d-shell" running at CPU 0 (see resource view) and it switched few times from user mode to system call mode in a short span of time. These few system calls are "sys_recvfrom" system calls occurring consecutively as shown in the Events view. The Unity-2d-shell[2] process is a component of Unity Desktop in Ubuntu that provides a shell interface to the system and "sys_recvfrom" is a system call that provides a common entry point for hackers to open a socket. Thus, this shows that a malicious hacker was trying to login into the system at this point of time.

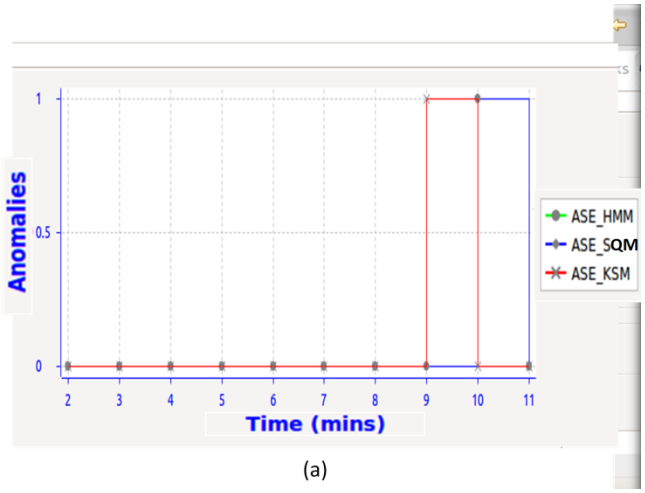

Figure 4: Detection of the attempt of an attack to bypass authentication mechanism in MySQL

In a similar manner to the sequence matching technique, the model of KSM also showed an anomalous alarm with the 42% system calls for file subsystem, 28% system calls of Kernel subsystem, 23% system calls of networking subsystem, 5% system calls of memory management subsystem, and the 2% system calls for the remaining subsystems. The HMM model did not point out a suspicious trace. HMM is a complicated but powerful algorithm and sometimes it converges to local minimum

[2] https://wiki.ubuntu.com/Unity2D;

rather than global minimum. It therefore requires additional tuning. However, SQM and KSM both pointed out the suspicious activity and it can be ascertained the trace was anomalous.

In addition to the attack on MySQL, we also executed other exploits on TikiWiki, Apache and brute-force attacks on SSH and FTP. The brute-force attacks remained unsuccessful; however, the attempts of the attacks were detected as suspicious. In summary, all the attacks were detected correctly and the false alarm rate remained less than 10%.

Thus, TotalADS can automatically detect the anomaly and facilitates in locating the origin of anomaly descriptively and visually. Such automated descriptive and visual analysis for software traces does not exist in the contemporary tools. This also answers our research question (Q2).

5. CONCLUSION

In this paper, we presented TotalADS, a new framework for integrating various anomaly detection techniques. The framework is built as a plug-in for Eclipse. It can be easily extended to support new techniques. TotalADS also allows the analysis, visualization, and diagnosis of traces for forensic purposes. TotalADS is built on the top of TMF, which is a powerful tracing framework that supports standard trace formats such as CTF.

During the development of the tool, we have experienced that extending open source technologies is harder than closed source technologies, due to the lack of documentation and tutorials. However, open source technologies have provisions for creativity that anyone can leverage, which is lacking in closed source technologies. TotalADS is currently evolving: additions of new techniques, creation of new interfaces, or improvements in existing ones are likely to occur.

We are currently working with DRDC (Defence R&D Canada) to conduct field experiments to test its usability in real environment. The tool is under review by Eclipse developer community and it will soon be available in Eclipse Market Place. Initial information about the tool is available on this link: www.ece.concordia.ca/~abdelw/sba/totalads/.

6. REFERENCES

[1] BugSense. [Online]. https://www.bugsense.com/

[2] Cittercism. [Online]. http://www.crittercism.com/

[3] Common Trace Format (CTF). [Online]. http://www.efficios.com/ctf

[4] G. Creech and J. Hu, "A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1-1, 2013.

[5] G. Creech and J. Hu, "Generation of a new IDS test dataset: Time to retire the KDD collection," in *In Wireless Communications and Networking Conference*, 2013, pp. 4487-4492.

[6] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff, "A sense of self for Unix processes," in *Proc. of the 1996 IEEE Symp. on Security and Privacy*, Washington, DC, USA, May 1996, pp. 120-128.

[7] S. A. Hofmeyr, S. Forrest, and and A. Somayaji, "Intrusion detection using sequences of system calls," *J. Comput. Security*, vol. 6, no. 3, pp. 151-180, Aug. 1998.

[8] W. Khreich, E. Granger, A. Miri, and R. Sabourin, "Iterative Boolean combination of classifiers in the ROC space: An application to anomaly detection with HMMs, Pattern Recognition," *Pattern Recognition*, vol. 43, no. 8, pp. 2732-2752, Aug. 2010.

[9] P. Lichodzijewski, A. Nur Zincir-Heywood, and M. Heywood, "Host-based intrusion detection using self-organizing maps," in *Proceedings of the 2002 Intl. Conf. on Neural Networks*, Honolulu, USA, 2002, pp. 1714-1719.

[10] LTTng. [Online]. http://lttng.org/

[11] Mahout. [Online]. https://mahout.apache.org/

[12] Massive Online Analysis (MOA). [Online]. http://moa.cms.waikato.ac.nz/

[13] Metasploit Pentration Testing Software. [Online]. http://www.metasploit.com

[14] MonogDB: A NoSQL Database. [Online]. https://www.mongodb.org/

[15] S.S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, "A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules," in *24th Intl. Symposium on Software Reliability Engineering*, Pasadena, 2013.

[16] National Vulnerability Database (NVD). [Online]. http://nvd.nist.gov

[17] Ossec: Host-based Intrusion Detection System. [Online]. http://www.ossec.net/

[18] L.R Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. of IEEE*, vol. 77, no. 2, pp. 257-286, Feb 1989.

[19] Samhain. [Online]. http://www.la-samhna.de/samhain/

[20] Strace. [Online]. http://sourceforge.net/projects/straceanalyzer/

[21] R Development Core Team, "R: A Language and Environment for Statistical Computing," *R Foundation for Statistical Computing*, 2011.

[22] TMF: Tracing and Monitroing Framework. [Online]. http://www.eclipse.org/linuxtools/projectPages/lttng/

[23] A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," in *Proc. of 3rd Intl. Workshop on Recent Advances in Intrusion Detection, LNCS*, France, Oct. 2000, pp. 80-92.

[24] W. Wang, X. H. Guan, and X. L. Zhang, "Modeling program behaviors by hidden Markov models for intrusion detection," in *Proc. of Intl. Conf. on Machine Learning and Cybernetics*, Shanghai, China, Aug. 2004, pp. 2830-2835.

[25] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proc. of 1999 IEEE Symposium on Security and Privacy*, Oakland, USA, May 1999, pp. 133-145.

[26] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. USA: Morgan Kaufmann Publisher, 2005.

[27] D. Y. Yeung and Y. Ding., "Host-based intrusion detection using dynamic and static behavioral models," *Pattern Recognition*, vol. 36, no. 1, pp. 229-243, Jan. 2003.