An Event-driven Architecture for Fine Grained Intrusion Detection and Attack Aftermath Mitigation

Jianfeng Peng, Chuan Feng, Haiyan Qiao, Jerzy Rozenblit
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721-0104, USA
jr@ece.arizona.edu

Abstract

In today's computing environment, unauthorized accesses and misuse of critical data can be catastrophic to personal users, businesses, emergency services, and even national defense and security. To protect computers from the ever-increasing threat of intrusion, we propose an event-driven architecture that provides fine grained intrusion detection and decision support capability. Within this architecture, an incoming event is scrutinized by the Subject-Verb-Object multipoint monitors. Deviations from normal behavior detected by SVO monitors will trigger different alarms, which are sent to subsequent fusion and verification modules to reduce the false positive rate. The system then performs impact analysis by studying real-time system metrics, collected through the Windows Management Instrumentation interface. We add to the system the capability to assist the administrator in taking effective actions to mitigate the aftermath of an intrusion.

1. Introduction

Network attacks are a fundamental threat to today's largely interconnected computer systems. Most of these attacks share the same characteristics: intrusion into computer hosts that have securities holes [1]. Unauthorized accesses and misuse of critical data on intruded hosts not only cause loss to personal users, but also pose a threat to the entire corporate network because in many cases the intruders use the compromised nodes to launch larger scale attacks. Firewall and antivirus packages are often insufficient in detecting and preventing all intrusions, especially attacks from insiders [2]. Efficient Intrusion detection systems thus are needed to form another important line of defense in the face of increasing vulnerability.

In the past, different types of IDS have been proposed and built. However, a study of currently existing IDSs reveals that most operate at a coarse grain level [3]. For example, Steven et. al. proposed an approach that uses sequence of system calls to identify potential threats [4]; Warrender proposed a similar approach [5]. Ghosh utilized return address information extracted from the call stack to generate an execution path for a program to detect anomaly [6]. While these IDSs use different feature representations of system calls, they treat events as an integral part and are unable to investigate internal characteristic such as executers, event objects, etc [7]. This directly affects detection accuracy. Another shortcoming of the coarse grain IDS is that reaction is only available after an event is completed, thus the system is unable to preempt a harmful operation before it completes.

To achieve more efficient intrusion detection, we need a more insightful understanding of the system's ongoing events. Fine grained event checking capability is an essential part of the architecture proposed in this paper. It is implemented using Subject-Verb-Object multipoint monitors. Any event that is taking place in the system is modeled using SVO structure. Alarms are triggered with respect to each element of the triple to achieve fine grained anomaly detection. The proposed architecture employs two databases that maintain metadata per user. The user metadata provides a basis for detecting deviation from normal user behaviors. The system metadata records real-time system performance metrics to facilitate alarm verification and impact analysis.

This paper is organized as follows: Section 2 briefly describes the SVO techniques used to model system events. Section 3 presents the proposed architecture. Section 4 discusses issues that are related to some of the function modules. Section 5 provides the DEVS Simulation results and some concluding remarks.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

2. Event Modeling Using Subject-Verb-Object Triple

In order to study the internal characteristics of an incoming event, we need a formal way to denote the following aspects: event executer, operation, and event object. In linguistic typology, such a structure is commonly called an SVO triple. Using this pattern, we can model an ongoing event with a triple that contains all the detailed information we are interested in.

2.1. The SVO Triple

Any event happening inside the computer host has its subject, which most of the time is a running process. For instance, typing a letter is often associated with a text editor; redrawing a client area is initiated by the parent window, while a *put* command in a sftp session usually comes from the ftp program. Similarly, we can describe the verbs and objects for these three events. Verbs of an event tell us what type of operation is performed on the objects. Objects of an event are normally hardware (peripheral devices, ports, etc) or software resources (files, drivers, libraries, etc) on the computer. At a higher level, these processes are owned by the current user, whose behavior is modeled by studying all the SVO triples when he is using the computer on a daily basis. By putting the subject, verb and object into a triple, we have a formal structure in the following way:

{	*MFC Window,*	*Redraws,*	*Client Area* }
{	*Text Pad,*	*Reads,*	*MyFile.txt* }
{	*FTP Program,*	*Opens,*	*Port 4567* }
{	*User Program,*	*Writes to,*	*Serial Port* }

…

We collect information of ongoing events using custom developed software tools. By dissecting an event into these three fields, we are able to perform fine grained analysis of the events. One may question the efficiency of this modeling technique as there are virtually infinite numbers of combinations that can happen when the computer is running. This problem is addressed by limiting the events to be scrutinized to a finite set of critical subjects, critical verbs and critical objects. Any event outside of this set will either pose no threat to the system or be a trivial threat that can be ignored. Fortunately, most of the events happening every moment on a computer do not belong to the critical event set (CES), thus we can focus on studying the behaviors of the ones that do belong to the critical event set as described in the following section.

2.2. The User-Configurable Critical Event Set

Since every user on a particular computer host has unique access privileges to resources, it is computationally very expensive to define a critical set that works for every user on every computer. Instead, it is necessary for the IDS to allow customizable critical set for each user. Depending on the nature of the data that reside on the computer and the actual role of the computer, this critical set can vary from a simple set that contains a few password protected files, to a much more comprehensive set of all files on C drive, local ports, and even hardware resources. In the current stage of our research, we focus on the critical set containing important files that need protection. The CES is defined as: {*, *, D:\EmployeeFiles*.doc}.

This critical set mandates that the IDS needs to monitor any process that attempts to perform any operations on any .doc files in the D:\EmployeeFiles directory. Consequently, any operation that is not accessing the data contained in that directory is ignored under the current configuration.

2.3. Adding Time Information

Each SVO, when saved into the user database, is tagged with a timestamp. The purpose of the timestamp is twofold. First, it adds an additional dimension to the event model and allows us to gain better understanding of the abnormality of current events. For instance, a particular file access operation is deemed normal during regular work hours; however, it is abnormal out of regular work hours. Secondly, timestamp makes it possible to perform temporal alarm fusion, which helps to reduce duplicated alarms.

By combining time information with an SVO triple, we have a mechanism that allows us to explore the following questions in a fine grained manner: who does what, to whom, at what time.

3. SVO Based Intrusion Detection Architecture

With the SVO modeling technique and custom defined CES described in section 2, we propose an architecture as depicted in Figure 1 to perform fine grained checking on all incoming events. When a new event comes in, it is intercepted by the IDS that runs in the background and sent into a multistage monitor. The monitor investigates the subject, verb and object by comparing them respectively to the normal behavior stored in the user database.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

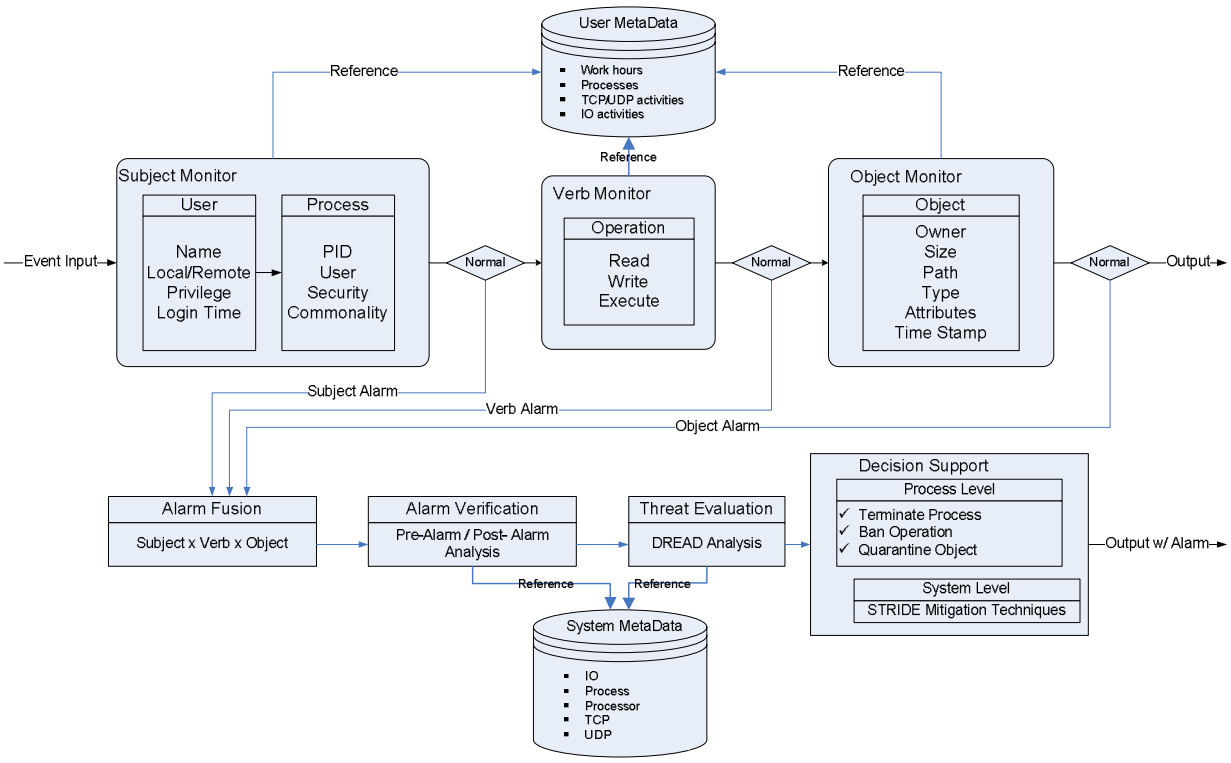

Figure 1 The Event-Driven Architecture

The user metadata database contains information that tells the IDS at what time which processes will normally take what kind of actions on what objects. A new event will trigger an alarm if it involves objects defined in the Critical Event Set, and if any element of the SVO triple deviates from normal behaviors. For instance, in our sample setting, all *.doc files in D:\EmployeeFiles directory are protected. Now if a process tries to modify a *.doc file in that directory, an Object alarm will be triggered first. The same event may also trigger Subject Commonality alarm if that process has never been observed before inside the current time window. In this case, the event violates multiple security rules and triggers multiple alarms, which are to be fused and verified subsequently.

The SVO based architecture has the following advantages. First, it doesn't require a priori knowledge of intrusions as needed in signature based [3] or system call traces based approaches [4]. Second, intrusion can be detected at the earliest time possible because the IDS does not need to wait until the operation has successfully been executed or damage has been caused. Third, an event is investigated at fine grain level so that more appropriate and effective reactions can be taken according to the nature of the intrusion.

Figure 1 shows all the essential components of the proposed architecture. These include the SVO based anomaly detection engine, alarm fusion module, verification module, threat evaluation module and decision support module. These modules address such problems as how to determine if an event is abnormal, how to deal with an alarm, what reactions should be taken when a threat is confirmed. The remainder of this section discusses the functionalities of these modules in more detail.

3.1. Anomaly Detection based on User Profiling

Intrusion detection techniques of IDSs can be categorized into two classes: signature-based and anomaly-based. Signature-based IDSs compare current events with known attacks and look for similarities, such as comparing a sequence of system calls to known attack patterns. This method has the major limitation of not being able detect novel attacks [3]. To overcome this, our proposed architecture uses an anomaly-based detection approach, which models normal behaviors and attempts to identify abnormal activities on the computer.

The precondition for such an anomaly detection IDS to work is to create a range of normal behaviors. In our system, this normal base is established through

an extensive user profiling process whose purpose is to build an in-depth knowledge of how the user uses this computer. The following information is derived: during normal usage of the computer, what processes take what kind of actions on what hardware resources or software objects?

To achieve this, we developed a software tool, SysMon as shown in Figure 2, which uses WMI to retrieve runtime system metrics. SysMon combines information from Spy++ event logs and records the results into a user profile database. More details regarding the database can be found in section 4.1. To model normal host behavior, both supervised and unsupervised learning algorithms can be applied.

Unsupervised learning algorithms take as input a set of unlabeled data and attempt to find intrusions contained in the data. It can be treated as a variant of the classical outlier detection problem and does not require the input data set to be fully normal. Outlier based anomaly detection algorithms cluster the data based on certain metrics and the points located on sparse regions are treated as intrusions. The unsupervised algorithms make two important assumptions about the data which motivate the general approach. The first assumption is that the number of normal instances dominates that of abnormal instances. The second assumption is that the abnormal instances are qualitatively different from the normal instances. The basic idea is that since the anomalies are both are rare and different from normal, they will appear as outliers in the data and thus be detected [8]. The clustering process scans through the data collected by WMI, and identifies normal instances and outliers.

From the results of the above mentioned clustering process, we have a clear knowledge of the user's normal usage of the computer. For instance, one can conclude that User A on the monitored computer has the following normal behavior with regard to file read operations in D:\EmployeeFiles directory during the regular work hours of *9am* to *5pm*:

Winword.exe,	*File Read Operations/sec,*	**.doc*	*[50,200]*

...

The above generalization gives an accurate indication of how a user normally uses his computer. Thus once a new event occurs, the anomaly detection engine will compare it to the existing profile of that user and determine whether the event falls into the range of normal behavior. Once a deviation is detected, the detection engine raises an alarm.

3.2. Alarm Fusion

Once an abnormal event happens, it is very likely to trigger multiple alarms at Subject, Verb and Object checkpoints. In order to minimize redundant alarms and condense alarms that stem from the same event into one integral alarm, an alarm fusion module is needed.

In the proposed architecture, a multi-level alarm fusion algorithm is used. The first one is a source preprocessing level, which synchronizes the information flow from different sensors to reduce data redundancy for further processes. The second level is alarm normalization, which transforms different alarms into a consistent set of scale. The third level is spatial alarm fusion, which fuses alarms from different anomaly detection monitors. The fourth level is temporal alarm fusion, which analyzes alarms within a certain time window and gives more useful intrusion information. Further details of the fusion mechanism can be found in [9]. After alarms are fused, they are sent to the verification module.

3.3. Alarm Verification

The task of this module is to verify the correctness of fused alarms in an effort to reduce false alarms. The verification module works by checking the normality of an event that has triggered an alarm. This includes checking the frequency of similar operations that have been performed before, as well as identifying the security level of the objects that the process is trying to access. While these are automated processes, the verification module also provides a human-computer interface that allows the system administrator to participate in the decision making process.

3.4. Impact Evaluation

A successful attack on a computer usually results in considerable impact on some aspects of its normal operation. Such impact includes disruption of critical services, undermined computation capability, increased network latency due to excessive outbound traffic and hardware resource exhaustion. To evaluate the impact of an attack, it is necessary to carry out a detailed comparison of system characteristics before and after an alarm is triggered. This is made possible by our real-time system performance monitoring tool, which collects system run-time performance data and saves them into a system metadata database as shown in Figure 1. The impact evaluation module queries the database about the following system metrics to find any differences before and after the event:

- Number of running services
- Processor time, queue
- Memory usage
- Network bandwidth, latency

3.5. Decision Support and Aftermath Mitigation

One of the major contributions of the proposed architecture is its capability to provide insightful information on current attacks and more precise counteraction with regard to the Subject, Verb or the Object. Depending on which elements of the SVO triple pose threats to the protected system, the decision support engine can automatically perform any one or a combination of the following three categories of actions:

- Ban current user
- Terminate operation
- Quarantine objects

In addition to the above three instantaneous actions, the decision support engine uses a rule based reason system that will investigate the nature of the attack, and take post-attack actions to eliminate security vulnerabilities and prevent the same type of intrusion from happening again. For instance, security level of the objects can be escaladed immediate to prevent future unauthorized access. A system administrator will be notified automatically. The administrator will often perform further investigation into the nature of the event and install patches to eliminate security holes. Howard proposed a variety of security precautions in building secure software [10]. Many of those techniques can be applied to unaffected systems on the same network to prevent the propagation of the attack.

The architecture also provides an interface that can report the current status to a network based fusion engine. The network based fusion engine collects information from each computer node, and correlates the data to perform higher-level situation analysis. Information provided by the interface includes everything that is needed to determine at what time, which user, by which process, is taking what kind of action against which object. With such fine-grained information, the network based fusion engine is able to take more effective actions to mitigate the aftermath of an intrusion such as terminating established TCP connections, closing ports, and isolating the affected host.

3.6. Performance Analysis

Once the IDS is installed on a computer, it runs constantly in the background to protect hardware/software resources defined in the CES. This adds to run-time computational overhead similar to antivirus packages. The overhead depends on the size of the CES. A smaller CES will have less impact on the system performance than a larger CES because the latter will involve checking on more events in run time. On computer systems that contain sensitive data, advantages of the SVO based IDS will greatly outweigh potential performance impact. The impact can be further reduced by introducing a security screening process that checks login attempts by any user out of normal operation hours.

4. Design Issues: Collecting User/System Data through WMI

The architecture proposed in this paper heavily relies on the capability to collect real-time information at both the system level and the process level. Microsoft WMI is a set of extensions to the Windows Driver Model that provides an operating system interface through which instrumented components can provide information and notification [11]. WMI includes real-world manageable components, available from the DMTF standards with some specific extensions that represent the various Windows components. To locate the huge amount of management information available from the CIM repository, WMI comes with a sql-like language called the WMI Query Language (WQL). WMI can be used to obtain data about your hardware and software by writing a client script or application, and data can be provided to WMI by creating a WMI provider.

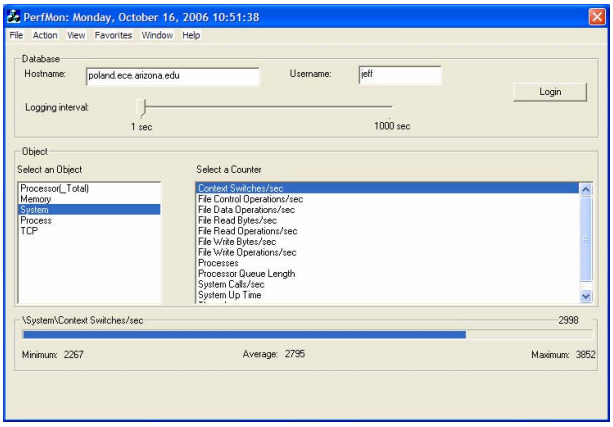

Figure 2 Real-time User Profiling & System Performance Monitoring Tool (SysMon)

Figure 2 is a screenshot that shows the SysMon tool we developed to collect system run-time information through WMI. The following is a snapshot of the information collected using the real-time user profiling and system performance monitoring tool that runs on our desktop systems:

ID	User	Usage	Counter	CurTime
11	1	1332	Processor(_Total) Interrupts/Sec	2006-09-20 17:08:04
12	1	16	Processor(_Total) Processor Time	2006-09-20 17:08:35
13	1	0	Processor(_Total) User Time	2006-09-20 17:08:59

…

5. Summary

To verify the overall effectiveness of the proposed architecture, a simulation platform has been built to simulate attacks and study the reactions of the IDS system. The simulation uses Discrete Event System Specification (DEVS) [12] to describe the proposed architecture. The simulation is described in detail in [7]. The initial experimental results are very promising. We are currently obtaining real world data on which a full verification of the proposed approach can be carried out. Our future research will focus on the mitigation of attack aftermath.

References

[1] B. Schneier, "*Attack Trends 2004 and 2005*", ACM Queue vol. 3, no. 5 - June 2005

[2] R. Chinchani, A. ILyer, H.Q. Ngo, S. Upadhyaya, "Towards a Theory of Insider Threat Assessment", Proceedings of International Conference on Dependable Systems and Networks, 28 June-1 July 2005 Page(s):108 - 117

[3] S. Axelsson, "*Intrusion detection systems: A survey and taxonomy*", Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

[4] S. A. Hofmeyr, S. Forrest, A. Somayaji, "*Intrusion detection using sequences of system calls*", Journal of Computer Security, Volume 6, Number 3, 1998.

[5] C. Warrender, S. Forrest, B. Pearlmutter, "Detecting intrusions using system calls: alternative data models", Proceedings of the 1999 IEEE Symposium on Security and Privacy, May 09-12, 1999.

[6] A. K. Ghosh, A. Schwartzbard, M. Schatz. "*Learning program behavior profiles for intrusion detection*", In Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring. USENIX Association, April 11-12 1999.

[7] A. Liu, C. Martin, T. Hetherington, Sara Matzner, "A Comparison of System Call Feature Representations for Insider Threat Detection", Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 15-17 June 2005.

[8] H. Qiao, J. Peng, C. Feng, J. Rozenblit, "Behavior Analysis-Based Learning Framework for Host Level Network Intrusion Detection", to be published in Proceedings of the 2007 IEEE International Conference and Workshop on the Engineering of Computer Based Systems, March 2007.

[9] C. Feng, J. Peng, H. Qiao, J. W. Rozenblit. "Alert Fusion for Intrusion Detection and Decision Support", to be published in Proceedings of the 2007 IEEE International Conference and Workshop on the Engineering of Computer Based Systems, March 2007.

[10] M. Howard and D. LeBlanc, "Writing Secure Code," Microsoft Press, 2nd edition, December 4, 2002.

[11] Microsoft, WMI: Introduction to Windows Management Instrumentation, http://www.microsoft.com/whdc/system/pnppwr/wmi/WMI-intro.mspx

[12] B.P. Zeigler and H.S. Sarjoughian, "Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models," Draft Version 3, 2005