# Pattern and Policy Driven Log Analysis for Software Monitoring

Ali Razavi

*Unversity of Waterloo*
*Department of Electrical & Computer Engineering*
*Waterloo, Canada*
*arazavi@swen.uwaterloo.ca*

Kostas Kontogiannis

*National Technical University of Athens*
*Department of Electrical & Computer Engineering*
*Athens, Greece*
*kkontog@softlab.ntua.gr*

## Abstract

*The component-based nature of large industrial software systems that consist of a number of diverse collaborating applications, pose significant challenges with respect to system maintenance, monitoring, auditing, and diagnosing. In this context, a monitoring and diagnostic system interprets log data to recognize patterns of significant events that conform to specific Threat Models. Threat Models have been used by the software industry for analyzing and documenting a system's risks in order to understand a system's threat profile. In this paper, we propose a framework whereby patterns of significant events are represented as expressions of a specialized monitoring language that are used to annotate specific threat models. An approximate matching technique that is based on the Viterbi algorithm is then used to identify whether system generated events, fit the given patterns. The technique has been applied and evaluated considering threat models and monitoring policies in logs that have been obtained from multi-user MS-Windows© based systems.*

## 1. Introduction

Large industrial software systems often consist of several collaborating applications that may be even deployed over local/wide-area networks. When these systems operate, events that are emitted from different and diverse sources are logged and stored for further analysis. Analyzing large log files is an overwhelming and complex task, especially when logged data pertain to event entries originating from different sources and applications. In this context, an interesting challenge is to devise techniques to define and denote specific patterns and consequently to design and implement analyzers that harvest the event logs in order to confirm whether these patterns appear in the event logs or not. For our purposes a *log entry* is a record of an event. Logged events can be classified as atomic or composite. An *atomic event* represents the occurrence of an action or a state, e.g., someone initiates the login procedure (action), or it is 12noon (state). A *composite event* consists of several atomic events matching a given pattern.

In this paper, we propose a monitoring framework that takes as input a threat model for a software system and produces a set of threat policy patterns that are denoted in specialized event policy language and ultimately as a Markov event state model. An approximate pattern matching technique can then be used to identify whether the events logged confirm or deny a specific policy pattern for a given threat model.

This paper is organized as follows. In Section 2 we discuss related work in the area of system monitoring and diagnostics. Section 3 discusses threat models and introduces a monitoring policy event language. Section 4 presents the proposed pattern matching technique and Section 5 concludes the paper and presents some pointers for future work.

## 2. Related Work

In the area of log analysis, Zhang in [1], proposes an analysis technique for log files with applications to both unit- and system-level testing. Similarly, Vaarandi in [2] presents a novel clustering algorithm for log file data sets which detects frequent patterns from log files,

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.81

and builds log profiles that identify anomalous logs. Denning presents a real-time intrusion detection system [3]. The system is based on the analysis of system usage to identify abnormal patterns of usage. In [4] an intrusion detection system is presented. The system is based on hidden Markov models that denote the normal operation of a system. An analysis technique detects intrusions by noting significant deviations from the model. The major difference with our work is that we model threats instead of normal behavior and then we attempt to verify that these threat patterns occur. Shieh et al in [5] present a pattern oriented intrusion detection system. The system is based on models that represent various intrusion patterns caused by the unintended use of programs and data. In [6] threats are modeled by Petri-nets and threat mitigations are modeled by Petri-net based aspects. In [7] an intrusion detection system that is based on the analysis of temporal orderings of system calls using deterministic finite automata.

## 3. From Threat Models to Event Patterns

### 3.1 Threat Models

Threat modeling is a process of analyzing and documenting a system's security risks. Threat models are usually encoded as Threat Trees. An example Threat Tree for a web application is depicted in Figure 1. Threat Trees are labeled ordered directed AND-OR trees. The parent node denotes a threat that is modeled and the children denote either sub-threats or events that must happen for the parent threat to be perceived as active and occurring. An introduction to Threat Modeling and Threat Trees can be found on [8].

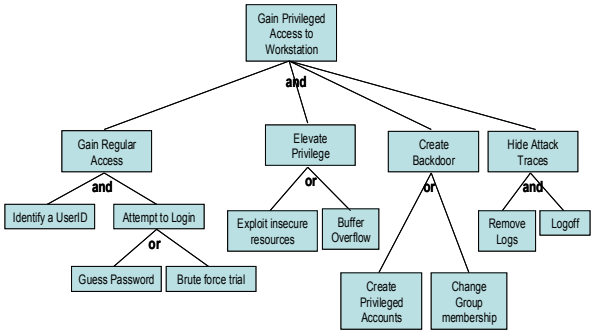


**Figure 1. A Threat Tree modeling actions required to take control of a server.**

Furthermore, threat models utilize threat trees to specify what is required for a specific threat to be realized. Any information in the order of events and the alternative forms of realization are usually presented in an informal format in the threat profile. While this serves its purpose as a security requirement document, it is not adequate for monitoring the behavior of a runtime system.

In this paper we propose the utilization of an abstract language which can be used to denote event patterns that complement threat models. This language provides the means to annotate threat trees with additional information pertaining to specific constraints and properties of events that can be associated with each node in the threat tree such as the event type, event ordering and, event timing. In this respect, we provide an initial guideline for the interpretation of logged events within also the context of a specific threat model or threat tree. The proposed event language is composed of atomic events, composite events and operators. The grammar of the proposed event language specified in EBNF is as follows:

*Script := Statement**
*Statement := Assignment | EventTypeDecl*
*EventTypeDecl := "EventType" Identifier ObservationList*
*ObservationList := "(" Observation( "," Observation)* "}"*
*Observation := Identifier ":" <PROB_LITERAL>*
*Expression := SeqExpr | TimedExpr | ConcExpr | ChoiceExpr | UnaryExpr | PrimaryExpr*
*SeqExpr := Expression ";" Expression*
*TimedExpr := Expression "After" <TIME_LITERAL> | Expression "@" <TIME_LITERAL>*
*ConcExpr := Expression "||" Expression*
*ChoiceExpr := Expression "+" Expression*
*Assignment := PrimaryExpr ":=" Expression*
*UnaryExpr := "!" UnaryExr | PrimaryExpr*
*PrimaryExpr := [Probability] (WildCardExpr | Identifier | "(" Expression ")" ) [ Process]*
*Probability := <PROB_LITERAL> "." Identifier*
*WildCardExpr := [Identifier] (("*"|"?") [Identifier])+*
*Process := "From" Identifier*
*Identifier := <ID >*

### 3.3 Observations and Atomic Events

An observation is a phenomenon that can be captured and recorded by the logging system. In other words, these are the concrete, application specific and observable events in the system.
An atomic event represents the occurrence of a single action, such as a login event, a function call or a file open operation. Each atomic event expression has a number of attributes such as the name of the event, the type of the event, the id of the process that initiated the

event, the timestamp of the event and a user-defined probability of occurrence.

### 3.4 Augmenting Threat Models with Event Patterns

Threat models are usually represented as Threat trees. Threat trees are ordered, labeled, directed AND-OR trees. The root of the tree represents the threat being modeled. The children of a node represent threats that must be achieved in order for the threat represented by the parent node to be achieved as well. An example of the mapping from Threat Trees to event expressions is illustrated in Figure 2.

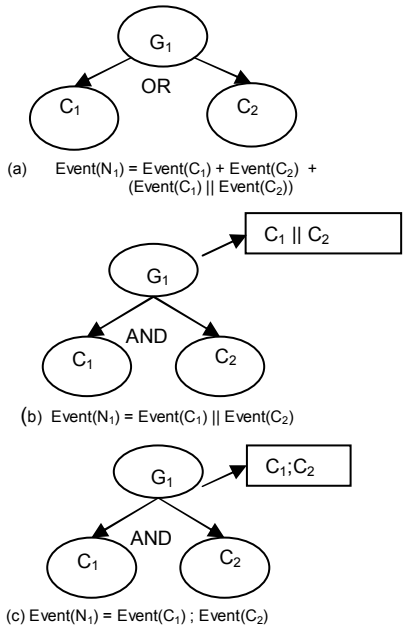


**Figure 2. Example mapping Threat Trees to event expressions**

More specifically, suppose we have a threat tree as in Figure 2(b) which denotes that in order for Threat $G_1$ to be materialized we need to have Threats $C_1$ AND $C_2$ be materialized. Threat trees on their own do not specify whether $C_1$ has to happen before $C_2$ or whether $C_1$ can happen in parallel with $C_2$. In this respect we provide to $G_1$ the simple annotation, in this example, of $C_1 \parallel C_2$. In this respect, Event(X) provides a class or a collection of low level logged events that are associated with X. Mappings form high level Threat Tree node types to low level log events and their frequencies are modeled a priori by the system administrator or the application vendor. Probabilities and frequencies can be updated as more information on the system's operational profile is gathered.

### 3.5 Mapping Event Expressions to Transition Models

Event expressions provide a textual representation of a Threat Model in a form where the threats and the actions are now represented as events that can be logged by the system's logging utilities. For example a Failed Login action that is considered in the Threat Tree, is represented now by a specific observable event that can be logged. Mapping event expressions to monitoring policy transition models can be defined in an iterative manner by considering the transformation rules that are illustrated in Figure 3. Once the transition model has been created the pattern matching process aims to identify the optimal assignment between the event transition model state model and the sequence of events logged by the system's logging utilities. The section below discusses in more detail the matching process.

## 4. Pattern Matching for Monitoring Policy

In this section we present the matching process that is used to identify important patterns in large log files. The proposed matching process contains the following steps:

1. System logs entries $e_{s1}$, $e_{s2}$, .. $e_{sk}$ are parsed so that an AST $T_{es}$ is created and event log description entries $E_{s1}$, $E_{s2}$, … $E_{sk}$ are obtained as nodes of such an AST. An event log description entry $E_{si}$, is an abstraction entity representing the actual event log entry $e_{si}$.
2. The event expression pattern $e_{p1}$ *op* $e_{p2}$ *op* $e_{pn}$ is parsed and a corresponding AST, $T_{ep}$ is created.
3. A transformation program based on the process discussed in the previous section, generates from $T_{ep}$ a state automaton leading to a composed Hiden Markov Model called the *Abstract Event Pattern Model* (AEPM) in which each state is a monitoring policy event expression statement and each transition link corresponds to the operators that can be inferred from the structure of the $T_{ep}$.
4. The Viterbi algorithm [9], [10] is then used to compute the probability that the composed HMM generates an event log description sequence (conversely matches an event sequence) $E = E_{s1}$, $E_{s2}$, .. $E_{sk}$.

The matching algorithm, terminates when all possible matches to reach a final state have been tried. The maximum length sequence of matched log entries $e_{s1}$,

$e_{s2}$, .. $e_{sk}$ that has the maximum matching probability among the sequences of log entries of the same length taken from the candidate sequences, is chosen as the result of the matching process. The Viterbi algorithm guarantees that all possible paths to a final state have been examined and that the best path, the one that maximizes the overall matching probability, is chosen.

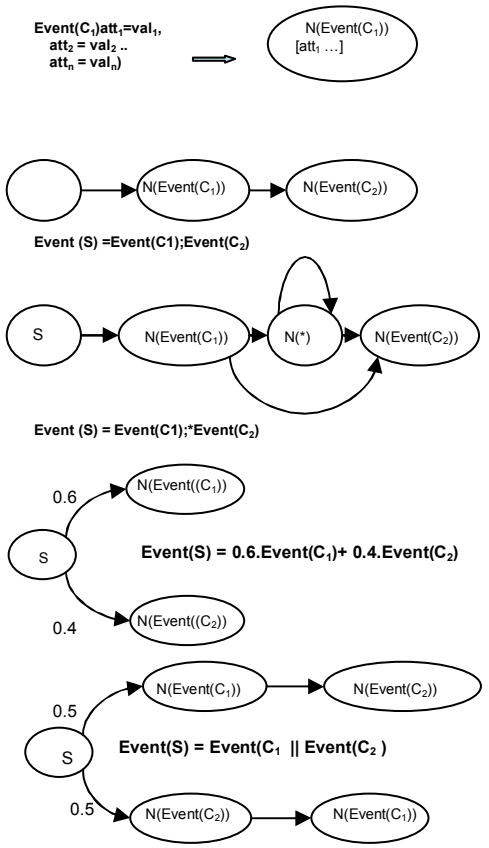


**Figure 3. Mapping event expressions to transition Models**

## 5. Conclusion

In this paper we presented a pattern-matching framework that can be used to identify such significant patterns in large event logs. The system is based on the idea that Threat Trees are annotated by abstract event patterns using an event expression language. The event expression is transformed into a state based Markov model. The Viterbi algorithm is then used to calculate the optimal alignment between the patterns and a sequence of logged events.

The benefit of the proposed system is on the flexibility it provides to the auditor or the software engineer to denote complex patterns that may relate to specific Threat Models for a given application. Future work for the proposed system includes the investigation of pre-processing techniques to limit the size and the complexity of the matching process. This work has been performed in collaboration with CA Labs and is supported by the Natural Sciences and Engineering Research Council of Canada.

## References


[1] J.H. Andrews, Y. Zhang, "Broad-spectrum studies of log file analysis", In Proceedings of 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, 2000, pp. 105 – 114.

[2] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs", In Proceedings of 3rd IEEE Workshop on IP Operations and Management (IPOM03), Kansas City, US., 2003, pp. 119 – 126.

[3] D. Denning, "An Intrusion Detection Model", In IEEE Transactions on Software Engineering, vol.13 No.2, Feb. 1987, pp. 222-232.

[4] S. Cho, "Incorporating Soft Computing Techniques Into Probabilistic Intrusion Detection System", In IEEE Transactions on Systems, Man and Cybernetics vol. 32, No. 2, May 2002, pp. 154 – 160.

[5] S.P. Shieh , V. Gligor, "On Pattern-Oriented Model for Intrusion Detection", In IEEE Transactions on Knowledge and Data Engineering, vol. 9, No. 4, Jul. 1997, pp. 661 – 667.

[6] D. Xu, K.E. Nygard, "Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets". In IEEE Transactions on Software Engineering, vol. 32, No. 4, Apr. 2006, pp. 265 – 278

[7] A. Kosoresow, S. Hofmeyr, "Intrusion Detection via System Call Traces", IEEE Software, vol. 14, No. 5, Sep. 1997, pp. 35-42.

[8] F. Swiderski, W. Snyder, "Threat Modeling", Microsoft Press, Redmond, Washington, 2004.

[9] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", IEEE Transactions on Information Theory, vol. 13, No. 2, Apr. 1967, pp. 260–269.

[10] G. Forney, "The Viterbi algorithm", Proceedings of the IEEE, vol. 61, No. 3, Mar. 1973, pp. 268–278.