A Survey of Intrusion Detection Analysis Methods

Justin Lee, Stuart Moskovics, Lucas Silacci
University of California, San Diego
jlee@cs.ucsd.edu, stuart@cs.ucsd.edu, lsilacci@cs.ucsd.edu
CSE 221 Spring 1999

1 Introduction

In modern computer systems, there is an increased need for secure operation. Modern systems tend to rely heavily upon networking and interoperation on public networks; the potential for misuse or abuse of these systems increases as accessibility increases. The complexity of modern systems makes detection of malicious activity difficult. In this paper we will give an overview of several intrusion detection systems. We will then describe two of the primary methods of intrusion detection: statistical and rule-based behavior analysis. We will discuss the implementation of these methods in current security systems and evaluate the strengths and weaknesses of each approach.

2 Intrusion Detection Systems

2.1 GrIDS (Graph-Based Intrusion Detection System)

GrIDS, developed at the UC Davis Computer Security Laboratory, is a system that collects data about activity on computers and network traffic between them. It aggregates this information into *activity graph*s that reveal the causal structure of network activity. By analyzing the characteristics of the activity graphs, GrIDS detects and reports violations of a stated policy [10]. This analysis is generally done through searching the graphs built for known bad patterns. A policy language to express acceptable and unacceptable behavior on the network is included with GrIDS so that an administrator can define policies in their departments.

The aim of GrIDS is to detect large-scale attacks such as *sweeps*, *coordinated attacks*, and *worms*. Sweeps occur when a single host systematically contacts many others in succession. GrIDS attempts to differentiate legitimate sweeps (e.g. audit sweeps by administrators, SNMP polling, etc.) from sweeps looking for vulnerable hosts. This is done by relying upon the highly regular nature of the legitimate sweeps in order to reduce the number of *false positives* (a misdiagnosis of an acceptable activity as an intrusion). Coordinated attacks are those that use parallel sessions in an attempt to obscure the nature of the attack, or to increase the attack's efficiency. To detect these types of attacks, GrIDS correlates sessions across several hosts and several disparate domains, building its activity graphs accordingly. Worms are generally known to be programs that propagate themselves across networks of machines, using the resources of those machines to infect still other machines. GrIDS easily detects worms due to the large amount of similar network activity and tree-like activity graphs they produce.

Since the main purpose of GrIDS is to detect network wide intrusions over a diverse network environment, one of its main objectives is to scale up for large network sizes. This is addressed through a method of hierarchical reduction for the activity graph construction. In the GrIDS model, an organization is modeled as a hierarchy of departments and hosts where each department has an intruder detection engine that builds activity graphs for that department. This graph is built using events from within the department. As graphs propagate upwards in the hierarchy, entire departments are represented as single nodes in the higher level graph, and cross-department activity is sent up to higher levels as well. In this way, each graph engine sees only the activity within its boundaries, and the amount of information sent to higher levels is greatly

reduced. Therefore, as a network expands to include more and more domains, the top levels of the hierarchy are not overly burdened with intractably large graphs.

Although GrIDS is designed to detect large-scale attacks or violations of an explicit policy, there are a couple key areas where certain attacks may go unnoticed. Unknown attacks that do not fall into the bucket of a "widespread attack" or a "worm" type of attack may easily escape the notice of a GrIDS based system. Also, the GrIDS "widespread attack" heuristics are very sensitive to the timing of individual attacks that makeup the "widespread attack." Because of this, a widespread attack that progresses slowly may not be diagnosed correctly.

2.2 NIDES (Next-Generation Intrusion Detection Expert System)

In 1983, SRI International began research on statistical techniques for audit-trail analysis that lead to IDES, or Intrusion Detection Expert System. Originally IDES only used statistical anomaly detection, but a later addition of the P-BEST expert system added a misuse detection component based on static knowledge [5]. IDES was later revisited by SRI, where they enhanced, re-engineered, and redesigned the system into NIDES. The NIDES system allows for multiple target machines to be audited from a single node that is dedicated to the analysis of audit data. One of the critical design decisions of NIDES was the decision to do the statistical analysis at the user level. In other words, NIDES determines whether user behavior as reported in the audit traces is normal with respect to past or acceptable behavior for each user [6].

Since NIDES is used to monitor several target host machines concurrently, the ability of NIDES to scale was watched closely by the system designers. The most important step taken towards this goal was to filter the audit data at each of the target hosts a NIDES system monitored. This is accomplished through an agent running on each monitored system called the *agen*. Agen, which stands for *audit data generation*, has a two-fold job: it converts all of the target host audit trail data into a generic format used by NIDES, and it preprocesses the audit data before forwarding it to the NIDES host. This preprocessing can greatly reduce the amount of audit data that the NIDES host must analyze.

Unfortunately, as the number of users increases, the amount of data that the NIDES host must analyze increases greatly. Since NIDES' statistical analysis is based on historical user behavior, a greater number of users requires a greater amount of storage and a larger amount of data to perform analysis upon. This has serious repercussions on NIDES' scalability. The system designers attempt to address this issue by describing a method for aggregating user statistics into groups, and then basing decisions on user-level anomalous behavior from those group statistics. Since this method is based upon the use of neural networks, the difficulty of retrieving the explanatory information behind such decisions makes this approach somewhat impractical [6].

The designers of NIDES felt that implementing the audit trail analysis and intrusion detection mechanisms on a machine separate from the system being monitored gave both performance and security advantages [1]. Their argument is that performing the analysis separately will keep from unnecessarily bogging down the target machine or otherwise affect its behavior, and that a standalone system is easier to make tamper-proof. This separation of analysis and target machines could effectively safeguard the intrusion detection system from any faults or performance degradation in the target hosts. Unfortunately, the entire system is not very fault-tolerant since an error on the NIDES host could potentially leave all of the target machines without any intrusion detection analysis engines.

2.3 EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)

Also developed at SRI, the EMERALD environment is a distributed scalable tool suite for tracking malicious activity through and across large networks [8]. EMERALD utilizes a hierarchically layered approach that includes *service analysis* for individual components; *domain-wide analysis* across multiple services in a single domain; and *enterprise-wide analysis* across multiple domains. At the lowest level, EMERALD uses independent distributed surveillance

monitors that provide *service analysis* for the individual components. EMERALD then builds upon these service-level monitors by introducing the notion of a *domain monitor* that is responsible for correlating reports from individual service monitors within the same domain. Enterprise-wide analysis is provided by once again correlating information from the previous level. The enterprise-layer monitors receive information from each of the domain monitors in the enterprise.

A monitor consists of three separate parts: a *target-specific resource object* which contains all the operating parameters for each of the monitor's components as well as the analysis semantics, an *analysis engine* which processes the target event stream, and the *resolver* which implements the response policy. Monitors are designed to enable the flexible introduction and deletion of analysis engines. Originally, these monitors started with a combination of signature analysis (*signature engines*) and statistical profiling (*profiler engines*) to provide complementary forms of analysis over the operation of network services and infrastructure.

The biggest advantages of EMERALD over previous intrusion detection systems are its scalability, its distributed correlation of information, its abstraction of host-specific information, and its improved fault tolerance. EMERALD improves upon SRI's previous work in NIDES by distributing the audit-trail analysis across all nodes instead of concentrating it on one host. The advantage of this approach is threefold: 1) Improved fault-tolerance (if a single analysis engine fails, other hosts are still protected by their local engines), 2) Improved scalability (adding more hosts does not add additional workload to a single analysis host), and 3) It allows domain-wide analysis engines to ignore host-specific information while performing analysis over a possibly heterogeneous domain. Once again, audit-trail information is reformatted and filtered before being handed "up" to higher-level analysis engines (as in NIDES). The designers of EMERALD also attempted to be very general in their approach. In particular, the decoupling of generic and target-specific concepts simplifies reusability and extensibility of components, and enhances integration with other data sources, analysis engines, and response capabilities [7]. A good example of this is that EMERALD has a profile engine that is completely separate from the mathematical algorithms used to assess whether events are anomalous or not.

3 Methods of Intrusion Detection

All of the systems described above include more than intrusion detection methods; they also contain approaches to respond to possible intrusions and a more complete framework for administrators to customize security policies and the extent of application and user monitoring. In this paper, we will focus on the methods used to identify system security violations and/or anomalous behavior. In this section, we discuss in detail the two main methods of intrusion detection: statistical analysis and specification-based methods.

3.1 Statistical Analysis Intrusion Detection Methods

The statistical approach is more of an exercise in anomaly detection; i.e., the statistical approach violations are defined as behaviors anomalous with routine observed system activity.

3.1.1 Philosophy

Statistical-based intrusion detection systems detect possible system intrusions by identifying departures from historically established normal behavior. They are concerned with long-term (historical) behaviors not exhibited in the short-term (recent) behavior, and the short-term behaviors not regularly shown in the long-term behavior. Some system designers have chosen to concentrate on the activities of users, while others have extended that paradigm to monitor application and network activity.

3.1.2 Implementations: NIDES

NIDES stores an audit trail containing program activity for each user. The stored statistics are the *profile* associated with a user; these profiles are updated periodically at an interval set by the system administrator. NIDES' statistical approach was to compare the long-term behavior of a user with the user's short-term behavior using their short-term and long-term profiles.

NIDES requires no *a priori* knowledge about the sort of behavior that would compromise system security [1]. The system is only concerned with extreme deviation from normal behavior. NIDES stores specific activity information (called "measures") about users of the system in their profiles. Each measure has a long-term and short-term behavior probability distribution. For example, a file access measure would involve both the recent file accesses for the short-term profile and the historical use of the file for the long-term profile. Other measures, such as CPU usage, are continuous measures in that they always have a current value. The usage of these measures would be assigned into a range of values called "bins".

NIDES uses a system parameter called a "half-life" that defines the number of audit records or days of audit activity that constitute short-term and long-term behavior. Long-term probability distributions normally use a half-life of 30 days. Using this setting, audit records 30 days old are valued half as much as the current records; audit records from 60 days contribute one-quarter as much weight, etc. Recent activity is always valued more than older activity; the long-term profile eventually phases out old behavior. A rule of thumb is that the effective length of the profile (in audit records) is approximately 1.5 times the half-life [1].

NIDES quantifies the difference between a measure's short-term and long-term profile as a numerical value *Q*. Each usage measure has a *Q* value. A large *Q* value indicates that the long-term and short-term profiles for a measure are very different from one another, while a *Q* value of zero means that the profiles match completely. A large *Q* value indicates suspicious behavior.

The system obtains the tail probability of obtaining a value for *Q* at least as great as the observed value. This probability is transformed (using a table look-up) into a value *S* with a half-normal distribution for each audit record. As an example, if the current value of *Q* were at the 80^{th} percentile (a tail probability of 20%), the value of *S* would be 1.28. A 90^{th} percentile *Q* would have an *S* of 1.65. High values of *S* correspond to unusual measures relative to the normal discrepancy between short-term and long-term profiles. The sum of the squares of the *S* values is combined into a value T^e, a summary of the abnormality of multiple measures [1].

For intrusion detection, a warning flag is raised whenever the short-term behavior is significantly different from the long-term behavior. For NIDES, audit records are anomalous at either the "yellow" or "red" level based upon the difference between the T^e value and historical values for T^e in the long-term profile [1]. As NIDES is a very large and complex system, how the system would respond to the red and yellow warnings is outside of the scope of this paper.

3.1.3 Implementations: Safeguard

NIDES was extended to profile the behavior of individual applications in a system called Safeguard. The statistical measures were customized to distinguish between the proper operation of a program and a trojan horse substitution. This was done because the NIDES environment included operating systems where users could compile and run their own programs, possibly masquerading as legitimate operating system programs. In Safeguard, an analysis was application-based, assigning a statistical score to the operation of each application. This score represented how far the application deviated from normal established patterns of operation [8].

3.1.4 Implementations: EMERALD

One of the major analysis targets of EMERALD is network gateway TCP/IP traffic. We will limit our analysis of EMERALD's statistical methods to those used in network traffic

monitoring. EMERALD tracks user activity via “measures” as in the NIDES system. Four classes of measures are defined for the statistical subsystem: categorical (i.e., host identity, port number), continuous (i.e., number of bytes transferred), intensity (i.e., rate of events per unit time), and event distribution (i.e., a distribution of the measures affected by recent events) [9].

EMERALD extended the work of NIDES and Safeguard so that components such as network gateways, proxies, and network services could be made subjects for analysis. Log files, packet analysis, and even special-purpose instrumentation of some services (i.e., FTP, HTTP, SMTP) were used as event streams for audit purposes. An event stream could be analyzed as a single subject or as multiple subjects. As an example, consider an event stream of dropped packets. The firewall might be the subject; a categorical measure would be the cause of the dropped packets; an intensity measure would be the rate of dropped packets during a period of time.

EMERALD generalized the ability to build abstract profiles for various types of activity. With regards to TCP-IP traffic, possible profiles included protocol-specific transactions, sessions between internal hosts and/or specific external sites, and application-layer-specific sessions.

Event records in EMERALD are generated at periodic intervals or in response to certain activities. For example, the system builds activity records based upon the content of IP packets. These records are coalesced into short-term and long-term profiles and scored for anomalous behavior as in NIDES.

Categorical measures in EMERALD monitor network traffic for anomalous activity. Some of the categorical measures are source/destination address, commands issued, protocols used, attempts to access restricted files, and error codes.

Continuous measures were also used when monitoring network activity in EMERALD. Some examples are the difference in time stamps between consecutive events from the same stream, number of packets, and number of kilobytes.

EMERALD’s intensity measure is concerned with the quantity of a certain type of traffic and whether that quantity is considered normal. Analysis of volume is used to detect suspicious traffic, such as denials of service or intelligence gathering. Abnormal increases in volume over a short period of time can indicate an attack or a physical degradation of the system. Long-term increases in event volume could indicate an effort to limit traffic flow. A SYN-attack against port availability or TCP/FIN messages indicating port scanning were also detectable using intensity measures [9].

EMERALD also uses a meta-measure termed “event distribution” that monitors which measures in the profile are affected by each event. As an example, an *ls* command during FTP will affect a directory measure but not a measure related to file transfer. The event distribution measure is not relevant for all event streams. Event records associated with network-traffic always affect the same measures for the network-traffic event, so the event distribution does not change. The event distribution measure is used to perform correlative analysis via an approach termed “Monitor of Monitors” [9]. Each monitor contributes to an aggregate event stream for the correlation monitor. However, these events are only generated if a recent behavior is declared anomalous. Intensity shown in the event distribution indicates a correlated attack.

3.1.5 Strengths

Statistical analysis methods allow the systems to adapt themselves to a diverse and possibly changing environments. For example, the NIDES team recognized that it is difficult to detect system intrusions based upon departures from normal behavior for individual users. As an example, consider the process of user logon. Some users have very well established logon behavior, such as logging on and off at very similar times every day (i.e., 9AM logon, 6PM logoff). However, some other users might have variable work hours, on some days working until early in the morning, other days not coming into work until late in the afternoon. The variable worker might also be a frequent traveler and could be logging in from different time zones. For

the variable worker, any sort of logon/logoff time can be considered normal. Therefore, storing the statistical profile for each user allows the system to detect abnormalities in user activity.

Statistical analysis can also be used to detect coordinated attacks that might not be noticed by only monitoring individual measures. EMERALD is concerned with a technique termed "doorknob rattling" where a mischievous user or an intruder attempts multiple and various techniques to compromise system security [9]. While one or a few instances of a violation may not be an intrusion attempt, an unusual number of suspicious system activities warrant suspicion. EMERALD's system is capable of working with open-ended values, including the capability to phase-out categories with long-term probabilities that fall below certain thresholds.

Data gathered during monitoring for intrusion detection can sometimes be used to detect other anomalies not related to system intrusions. For instance, EMERALD is capable of using its continuous measures to monitor the health of the system and network. A sudden drop in traffic can point to a hardware or system failure.

3.1.6 Weaknesses

Statistical approaches of intrusion detection can possibly turn up false positives when monitoring for anomalous activity. Since these systems are tuned to detect unforeseen threats, anomalous but innocuous behavior might trigger a statistical-based warning [7].

The NIDES audit data wasn't considered well suited for the authors' analytical purposes, and they desired different sources of data with greater abstraction. The authors also desired to use higher-level data such as database management systems, but the security policies of the database management systems permitted what was already closer to normal behavior (thus not providing much insight into actual intrusion detection).

The NIDES statistical detection system did not scale well to distributed and networked environments for the two following reasons: 1) For analysis, the measures needed to be treated in their entirety, and 2) The results weren't stored in a format that was useful at a higher-layer. The authors also learned that profiling functionality was more effective than profiling individual users. They concluded that their statistical analysis might be useful for whole systems and subsystems, including servers and routers [7]. This flaw was addressed in EMERALD, enabling the newer systems to overcome the previous weaknesses.

It is theoretically possible to train a statistical system to allow bad behavior. Since the system stores long-term and short-term profiles, introducing possibly malicious behavior into a routine activity over a large amount of time would show that the behavior is routine and allowed for a certain user or application. This flaw might allow malicious behavior to proceed without being flagged with a warning by the statistical system.

3.2 Specification Intrusion Detection Methods

The second major method for defining malicious system activity is the specification based (also referred to as a rule-based or signature based) approach. The general idea is to map system activities to a set of specifications that define the difference between allowable and undesirable behaviors. In a purely specification-based approach, the system implementers define which behaviors raise violations.

3.2.1 Philosophy

There are two basic ways to conceptualize a specification-based intrusion detection algorithm. The algorithm can be designed in what we will refer to as an inclusionary fashion; that is, the algorithm will attempt to map system activity against pre-defined malicious behavior patterns. This approach is used in the specification-based modules of EMERALD and NIDES. These inclusionary modules have heuristics that are used to actively identify specific types of

attacks. For example, the EMERALD system's eXpert engine has separate rule sets designed to recognize login password hacks, buffer overruns, and SYN flood attacks [5].

Alternatively, an algorithm can be designed in an exclusionary fashion. In other words, the system will have a set of rules that describe allowable behavior; any behavior that does not comply with the rules is considered a violation. This is the strategy that has been suggested and implemented by Levitt, Ruschitza, and Ko with the DPEM (Distributed Program Execution Monitor) system [4]. They developed a specification-based security monitor for UNIX which protected key system utilities by parsing audit trails and flagging event streams that did not qualify as proper instances of their specification grammar.

Obviously, there are strengths and weaknesses to both inclusionary and exclusionary approaches. The inclusionary model can be tuned to catch known attack patterns very well. Unfortunately, a monitor using only the inclusionary form of intrusion detection will be vulnerable to new methods of attack that it has not been pre-programmed to deal with. In an exclusionary model such as DPEM, this problem can be avoided. The DPEM monitor has the potential to flag previously unknown attack methods if these methods in any way attempt to deviate from acceptable system activities. However, the exclusionary model can potentially have a problem that is conceptually opposite to that of the inclusionary model; it can be too restrictive and flag activities that are not malicious in nature. DPEM addressed this problem by localizing their specifications to a handful of vulnerable system utilities [4]. Unfortunately, we must know quite a bit about how a malicious attack might be orchestrated against the system (as in the inclusionary model) before we can build an effective monitor. The difference in this case is that the foreknowledge about attacks is in the form of restriction localization, as opposed to attack detection heuristics.

3.2.2 Implementations: Representations

The specification paradigm maps well to the concept of grammars and parsing engines. Therefore, specification-based monitors are typically built as parsing engines. The input is the audit trail or network traffic pattern; the grammar is the set of specifications which describe allowable (exclusionary) or illegal (inclusionary) behaviors; an unsuccessful (exclusionary) or successful (inclusionary) parse is viewed as illicit activity and subsequently raises a violation. GRIDs and LaSCO (another graph based system developed at UC Davis [3]) expand the power of this paradigm by formalizing graph-based grammars. Instead of merely providing a yes/no type of analysis, graph-based grammars have the capability of building behavior patterns which can then be compared against a database of known attack patterns and subsequently flagged.

3.2.3 Implementations: Real time vs. offline monitoring

As with statistical representations, rule based representations have a choice of monitoring in real time or offline. Real time monitoring has the benefit of reducing the potential damage with a swift remedial response once a violation is flagged. Unfortunately, this comes at a higher overhead cost. In the case of specification-based monitors, algorithm efficiency is a big issue because parsing engines are by nature very CPU intensive. A multi-user, multi-process machine generates a very large amount of activity data; system engineers must ensure that if their security mechanisms operate in real time, these security mechanisms should not significantly impact the overall performance of the system.

Another possible problem concerns locality. More specifically, a naïve implementation of a real time specification-based approach might assume that a prohibited sequence of system events has some sort of localization according to time (i.e., occurs closely together in time) or user (i.e., is initiated by a single user). An example of this situation is when a normal user changes his password while the superuser is editing the password file, leaving the password file in an inconsistent state [4]. Statistical profiling systems and offline rule-based systems are better

suited to detect these types of attacks because they are not bogged down by performance considerations, and therefore can look at a larger, more expansive profile or audit trail.

The DPEM system is an example of a real time, rule-based intrusion detection system. It solves the performance issue using two strategies. DPEM rigorously filters audit trails with modules referred to as trace collectors before passing the audit trails on to the parsing engine. The other way DPEM achieves good performance is by concentrating only on system actions and ignoring data values. This lack of data monitoring is a weakness explicitly acknowledged by the authors of the system [4]. DPEM addresses the problem of synchronization based violations by splitting up its parser into processes that each follow any related streams of activity simultaneously.

Alternatively, the system engineer can choose to have his/her monitor work offline, grabbing cycles when they are available. This has the advantage of allowing more freedom in terms of performance constraints. The system engineer can create a more expressive grammar to describe system violations without worrying as much about the overhead this induces. The obvious downside to this approach is that a violation may not be sensed until hours or days after the violation has occurred, depending on at what intervals the monitor is directed to run.

EMERALD's eXpert signature analysis subsystem uses the offline approach. The eXpert module supports a highly expressive inclusionary grammar (P-BEST). In addition to a powerful grammar, the eXpert system has the added benefit of being able to analyze extremely large audit collections (on the order of several days) to detect intrusion attempts that could be spaced out over time and among users [5]. This strength is also a potential weakness, in that eXpert's preferred analysis periods of once every one or five days means that remedial actions and/or network trace attempts are much less effective.

3.2.4 Strengths

The rule-based approach has a number of features that make it an attractive alternative when designing an intrusion detection system. With its foundations in descriptive grammars, rule-based systems fit well with the principles of abstraction. An intrusion detection monitor designer can implement the core monitor facilities and a basic grammar, and then let individual system administrators tailor the monitor to the needs of their own systems. These systems also scale well; as seen with the GrIDS system, the specification-based approaches can be easily adapted to local, LAN, or WAN environments [10].

The relationship between specification approaches and grammars has the added benefit of easing implementation. Building parsers is a well-defined and well-explored area of software engineering. Monitor builders can leverage off of this by using the easily available tools and algorithms that have already been designed to solve the parsing problem.

Conceptually, the specification approach is also appealing because it simplifies the test operation for deciding whether or not a set of events constitutes a violation. If the event stream does not pass the parser test, it is a violation. System administrators do not have to deal with subjective and imprecise threshold values like they do with statistically based systems. Of course, this sense of absolute definition can be misleading if the specification is poorly constructed, or if the grammar is not capable of expressing a monitor that correctly flags the offending system activities.

Finally, the rule based approach has the benefit of taking advantage of the knowledge system administrators have about possible attacks. Administrators can use these systems to fine tune their systems against well-known attack patterns, and plug all of the security holes they are aware of.

3.2.5 Weaknesses

Of course, the rule-based approach also has its weaknesses. The most obvious of these is the very real possibility that the specifications do not properly define the difference between allowable and illegal behaviors. In the case of an inclusionary specification system, if an attack has not been defined in the specification database, the attack will not be considered a violation. The exclusionary method can be, as mentioned previously, too restrictive, and the techniques used to get around this problem indirectly give exclusionary methods the same weakness as inclusionary methods.

Pure statistical approaches are strong in that they can adapt to changing trends in system usage. Specification approaches are in contrast much more rigid; a rule set must be explicitly modified by system maintainers to adapt to a different working environment.

Finally, rule based approaches tend to be costly in terms of the system resources they require to function. Parsing is a very resource intensive activity; rule complexity can have a direct tradeoff with performance [8]. In other words, steps must be taken to ensure that event monitoring is effective, yet does not absorb too many system resources, especially with monitors that work in real time.

4 Conclusion

Statistical and rule-based methods each have their own benefits and drawbacks. The two methods have the interesting property of complementing each other; i.e., the weakness of one is a strength of the other. This suggests that an engineer designing a security subsystem for a distributed system should consider incorporating elements of both methods to ensure maximum coverage. An interesting direction for further research would be to come up with a method of intrusion detection that combines the adaptability of statistical methods with the embedded knowledge and user-optimization capabilities of the rule-based approach. One way this can be achieved is by using AI techniques such as genetic programming. Instead of filling out rules or specifications for a module to use when monitoring a system, Crosbie and Spafford use example event sets to ‘train’ their agents about what defines a violation [2]. Even after the training sets are processed and the agents are monitoring real system events, the agents have the ability to learn from observed behaviors and adjust their sensing algorithms accordingly. Another possibility for combining statistical and rule-based methods would be through the use of stochastic grammars. Statistical information can be used to adjust the probabilities of individual rules of the specification grammar, giving the monitoring system the ability to adapt to its environment after the initial specifications have been implemented by the system administrator.

5 References

[1] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes, “Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)”. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, May 1995.

[2] M. Crosbie, and G. Spafford, “Active Defense of a Computer System using Autonomous Agents”. Technical report, COAST Group, Dept. Of Computer Sciences, Purdue University, February 15, 1995.

[3] J. Hoagland, R. Pandey, and K. Levitt, “Security Policy Specification Using a Graphical Approach”. Technical report, UC Davis Computer Science Computer Security Laboratory, July 22, 1998.

[4] C. Ko, M. Ruschitzka, and K. Levitt, “Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach”. *1997 IEEE Symposioum on Security and Privacy.*

[5] U. Lindqvist, “Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)”. *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, Oakland, California, May 9-12 1999.

[6] T. Lunt, “Detecting Intruders in Computer Systems”. *1993 Conference on Auditing and Computer Technology*.

[7] P. Neumann and P Porras. Experience with EMERALD to Date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara California, April 11-12, 1999.

[8] P. A. Porras and P. G Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353-356, Baltimore, Maryland, Oct. 7-10 1997. National Institute of Standards and Technology/National Computer Security Center.

[9] P. Porras and A. Valdes, “Live Traffic Analysis of TCP/IP Gateways”. *Proceedings of the 1998 ISOC Symposium on Network and Distributed Systems Security*.

[10] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, "GrIDS - A Graph-Based Intrusion Detection System for Large Networks". *The 19th National Information Systems Security Conference.* 1996.