# A Case Study on Constructing a Security Event Management System

Vijay K. Gurbani, Debra L. Cook, Lawrence E. Menten, and Thomas B. Reddington
Security Technology Research Group,
Bell Laboratories, Alcatel-Lucent
{vkg,dcook,lmenten,treddington}@alcatel-lucent.com

**Abstract:** We define *Security Event Management* (SEM) as the ability to analyze the information arriving as discrete events from various network services in order to determine whether the network, or a portion of the network, is in the process of being compromised *and* to undertake evasive action to mitigate the attack. In practice, a SEM system can be viewed as the collection of tools, technologies and policies related to presenting a security-specific view of the network at all times We describe our work in constructing a security event management system using a mix of open source and internally developed software. Our results in constructing such a system and lessons learned during the process are presented in this paper. We also outline an agenda for future research in this area.

## 1 Introduction

For the purpose of this paper, we define Security Event Management (SEM) is the ability of a network to analyze and interpret discrete events in order to enable better security assessment *and* undertake appropriate remedial action (this characterization is more expansive than the traditional held definition of a SEM as a detection only step; mitigation is provided by other means.) SEM capability is necessary in today's networks because network security is notoriously difficult to manage due to the following four reasons: First, network security has been largely delegated to individual hosts and applications executing on them; this leads to a plethora of points to monitor when a security breach occurs. Second, the hosts and applications executing on them typically have limited communications with other security products. As a result, there is substantial difficulty in detecting large scale network-based attacks. Third, most network security is reactive, not pro-active, and as a result is done after an attack has occurred and is well underway. Finally and most importantly, network security tools today do not lend themselves to adequate situational awareness by failing to provide an integrated network view regarding the state of the network.

Figure 1 depicts a bi-directed cyclical graph that can be used as a model for securing the network (we consider the graph bi-directed because for every edge in the graph, the reverse edge also belongs to the graph.) The graph consists of three vertices: Prevention, Detection and Response; the edges are labeled according to the functions they provide between the vertices.

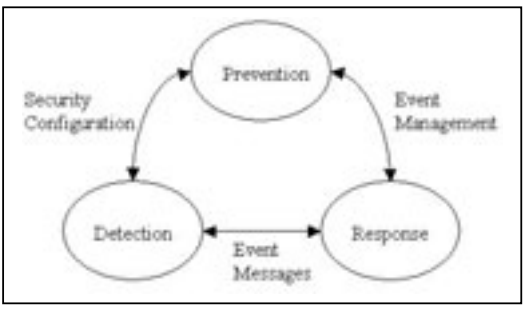


Figure 1: The Security Triad

The graph forms a feed-forward system (see Figure 2) where the output from a state is used to drive the behavior of the next state. For example, if a host detects that it is undergoing a port scan (current state: Detection), it will transition to the next state (Response) by issuing an event message. In this state, the host uses event management request („Block source IP address X“) to transition to the next state (Prevention), where the event management request is subsequently converted to an event management directive enforceable locally or at a router or firewall to prevent the host from being scanned further.

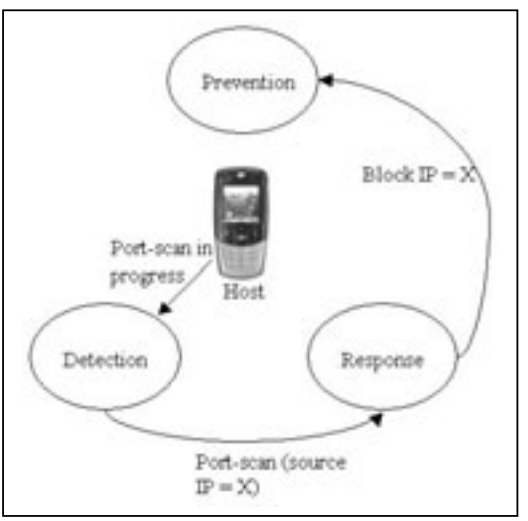


Figure 2: Feed-Forward System

Figures 1 and 2 essentially describe a SEM system. SEM is the „glue“ that allows the security triad to work together in an integrated fashion. A typical SEM architecture (see Figure 3) consists of three layers – a user interface, an event correlation engine, and event collectors. The user interface provides a graphical view of the situational awareness to the network the operator, and depending on the specifics of the SEM system, may be used to specify policies (authentication, access control, etc.) on the individual hosts and applications in the network.

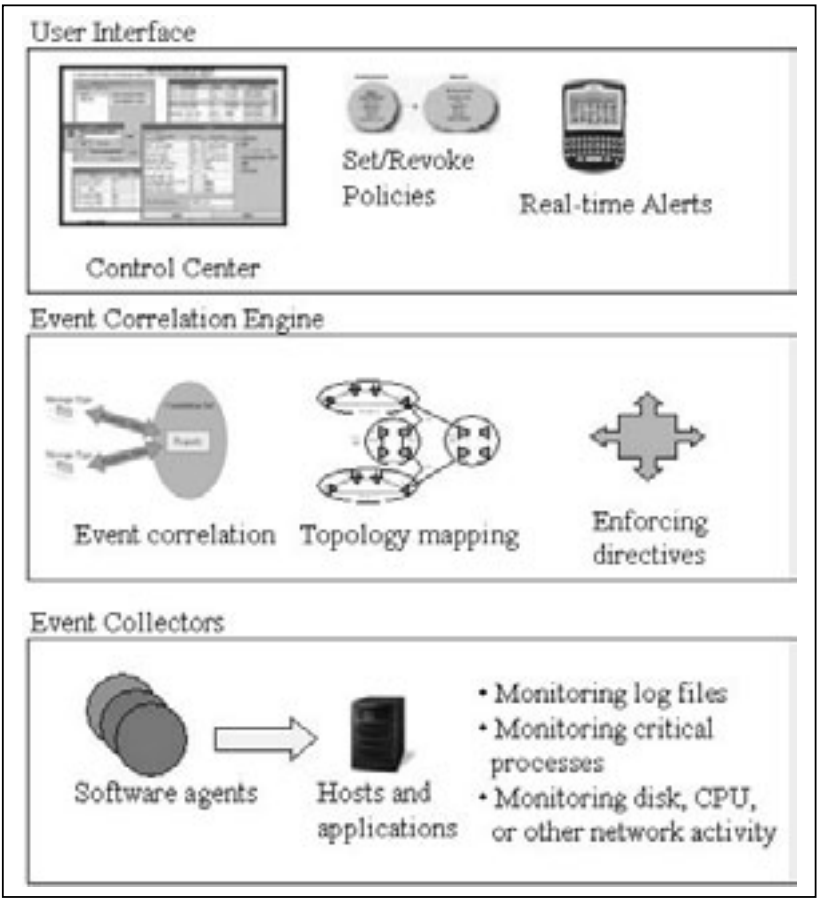


Figure 3: A Prototypical SEM Architecture.

The second layer, the event correlation engine, corresponds to the „Response“ state in Figure 1: it is concerned with real-time response that quarantines affected parts of the network. A robust correlation engine will require a meta-language to specify and capture events; „data fusion“ capability to describe the overall behavior of the attack based on discrete events arriving into the engine; semi-automatic response to thwart

attacks before they infect the entire network; and the ability to securely execute commands on other network elements.

The third layer, event collector, corresponds to the „Detection“ state of Figure 1. Event collectors are software agents co-located on hosts or with applications. When an agent detects behavior that is contrary to the normal operation of the host or application, it informs the event correlation engine of this occurrence. The event collectors agents have to be tuned to a particular application, for instance, an agent for a web server may monitor the log file and intimate the correlation engine of abnormal behavior (such as rapid requests for accessing resources coming from the same IP address, an attempt to access a resource known to be vulnerable, etc.)

The remainder of this paper describes our work at creating a SEM system using open source and internally developed software. Section 2 reviews the existing literature on SEM systems and Section 3 contains the system details of our SEM system. This is followed by a series of „what-if“ scenarios that tests the system by launching attacks on the network. Section 5 details observations and lessons learned from building a SEM system; Section 6 outlines an agenda for future research in this area.

## 2 Related Work

The concept of and the need for a SEM system has been recognized in the industry; there are a number of white papers on SEM systems and SEM implementations from commercial companies. However, for our continuing research outlined in Section 6, we require a SEM framework that we could control on a level that would exceed the control offered by a commercial implementation. For instance, the ability to swap the event correlation engine, or the ability to extend the SEM system through new software agents that are closely coupled to leverage the correlation engine or the topology mapping engine would be immensely beneficial. Primarily to gain such fine grained control of the system, we do not consider commercial implementations further in this paper.

Liu et al. [11] describe a SEM framework constructed using case-based reasoning. Their framework consists of four modules that bear similarity to the components we used in constructing our SEM system. The „Data Collection“ module uses agents to collect raw data by analyzing event log files and security log files . The „Data Standardization and Aggregation Module“ cannonicalizes the data from multiple sources into a format that the next module, the „Event Correlation and Scenario Analysis Module“ can operate upon. Finally, their „Quantization and Output Module“ uses data-mining techniques to compute the threat level of an event. Ertoz et al. [8] describe MINDS – Minnesota Intrusion Detection System, a project that also uses a suite of data mining techniques to automatically detect attacks against computer networks and systems. The traffic from the University of Minnesota is monitored using snort, an open source network intrusion detection system. The data is filtered and fed into the MINDS system, where the known-attack detection module detects attacks that correspond to known signatures. The remaining data is fed into an anomaly detection module, which assigns a score that reflects how anomalous the data is compared to

normal network traffic. Highly anomalous attacks are further categorized to create new signatures and models for emerging attacks; thus forming a feedback loop.

There is tremendous activity in the techniques associated with a SEM system, such as intrusion detection and data mining techniques to determine root cause analysis. Duan et al. [7] and Sekar et al. [15] suggest techniques to enhance intrusion detection systems to minimize the false alarm rate. Ning et al. [13] present a method for constructing attack scenarios through alert correlation and Julisch [10] observes that a few dozen persistent root causes accounts for over 90% of the alarms that an intrusion detection system triggers. They propose a novel alarm-clustering mechanism to identify the root causes of an alarm. Devitt et al. [6] uses the topological proximity approach that exploits topological information embedded in alarm data to filter out alarms that are not plausible from the point of view of the network topology.

In a SEM system, a standard protocol and an associated set of application programming interfaces (APIs) between the event correlation engine and the specific network component reporting the event is required. Debar et al. [5] have defined the Intrusion Detection Message Exchange Format (IDEMF) protocol that describes data formats and exchange procedures for sharing information of interest to intrusion detection and response systems and to the management systems that may need to interact with them. The Intrusion Detection Exchange Protocol (IDXP), is an application-level protocol for exchanging data between intrusion detection entities [9]. IDXP supports mutual-authentication, integrity, and confidentiality over a connection-oriented protocol. The protocol provides for the exchange of IDMEF messages, unstructured text, and binary data. However, it is unclear how widespread the use of IDMEF and IDXP is; the protocols are considered experimental in nature.

More recently, Mitre has developed the Common Event Expression (CEE) which addresses the problem of vendors and products employ varying logging practices such as using inconsistent formats and terminology when describing events [12]. This alleviates the significant burden to analysts and products in normalizing the vast quantities of heterogeneous log records, allowing for aggregation, correlation, and further processing. At this time, there aren't any published papers on CEE to evaluate it in depth that the authors are aware of.

The work we present in this paper is orthogonal to intrusion detection, data mining and event reporting techniques. Indeed, the module in our SEM framework that correlates the event can be augmented (or even replaced) to take advantage of the results obtained by other researchers working in this area. In the same vein, even though the protocol used in our SEM to report events of interest to the event correlation engine is not standardized, it could in the future be replaced with a widely deployed one should such a protocol become available.

## 3 System Design and Architecture

Security threats have an exploitation cycle that permits a SEM system to act early enough to be effective; time is of essence during an exploitation cycle: if the SEM system is slow to react, the security breach would have already occurred. Our goal in constructing the SEM system was simple: we wanted to detect the discrete events as they build up to a security attack and acting consistent with the security policy, undertake mitigation actions before the attack could have an impact on the network resources.

### System Components

To construct a SEM system that met this goal, we needed a strong event correlation engine. There are several examples of scalable event-based middleware: CEA [1], Sienna [2], JEDI [4] and TOPSS [3]. However, the common drawback with all of these frameworks is their lack of strong correlation capabilities. Correlating discrete events from multiple sources of information into one view is an absolute requirement and a cornerstone of an SEM system. From this correlation arises the ability to track the security event and automatically respond through rules that are triggered as certain conditions are met. A triggered rule results in an automatic response to contain the attack and alert the security operator so that human-decision making can be subsequently engaged.

In the end, we constructed our SEM system using a in-house developed high-performance fault manager as the core of the system to achieve distributed platform with good alarm thresholding and suppression features, a language for creating rules to correlate events of different types and from different sources and with network topology information from an integral topology database. The platform also provided flexible interfaces for collecting event information from network devices and good facilities for the customization of event collection and the reformatting of events into alarms. Another advantage of the in-house software was that it was integrated into a browser, complete with color-coding of alarms and the ability to send email or page an operator. With this piece of software, we essentially had Layers 1 and 2 (User interface and event correlation engine, respectively) from Figure 3 in place.

In order to collect the events, we used a variety of in-house and open source tools. For HTTP and SSH common log file analysis, we used an in-house log file analyzer. This analyzer attached itself to the log files to be monitored and if it detected an anomaly in the normal operation of the HTTP or SSH daemon, it would send a number of denunciations to the correlation engine. For HTTP, the analyzer software maintained a list of over 2000 known web server vulnerabilities catalogued by Common Vulnerabilities and Exposure dictionary (http://cve.mitre.org). If the software detected that a cracker was attempting to access a resource listed in the CVE database, it would immediately issue a denunciation to the correlation engine.

In addition to the static list of vulnerabilities, the analyzer is also tuned to dynamically perform statistical analysis on the requests to find attack patterns. For HTTP requests,

the statistical analysis observes certain characteristics, such as inter-arrival time, errors generated, and links accessed. Some other measures like bandwidth utilized can also be factored in to compute a weight for the IP address corresponding to the connection at the server. Depending on the value of the weight, the connection could get dropped, experience differential service, or allowed. Some weighting factors are exponential, for example, if a connection is using more than 75% of the bandwidth, it will get dropped. For SSH requests, the analyzer software looks for failed attempts from the same source IP address; if it detects such a pattern, it will issue a denunciation for that IP address.

We used snort for an intrusion detection system (IDS). Snort was configured to write messages to the system's */var/log/messages*. A Perl script constantly monitored the file and forwarded the messages of interest to the correlation engine as a security event. We also wrote a file signature verification software that was used to track changes to files and the addition or removal of files from a user's directory. Finally, we used a firewall to protect access to the core hosts in our network (see Section 4 for a network diagram of our laboratory setup) and as an additional security event detector in our network. Note that although we used a firewall developed internally and sold commercially, any other vendors firewall could be used. The firewall that we used provided session establishment rate limiting, traffic rate-limiting, and good detection and alerting features for invalid IP address and IP header content, and detection and alerting for TCP state violations. We used these features to provide an additional source of security events for the correlation engine. Collectively, the log analyzer software, the snort Perl logger and the firewall logger act as data collectors with reference to Figure 3.

**Architecture**

Figure 4 contains the SEM framework corresponding to the discrete parts we describe above. There are three datasources where external events are fed into the system. The events enter into the system in a canonical format, from where the data collector distributes the events to a system logger and a data distributor. The data distributor can be programmed to distribute these events to operation support interfaces or to other external destinations. The data distributor also sends the data to a correlation engine, which accepts the events and processes them according to rate, time, group or value thresholds to create specific alarms or to send a rule to another managed element (firewall, for example) for limiting (or allowing) access to an IP address or a range of IP addresses.

## 4 Using the SEM Framework

We constructed a laboratory network depicted in Figure 5 to test our SEM system. The network consists of four machines behind a firewall. Machine A is the primary attack machine that the cracker will try to break into. Machine B runs snort in promiscuous mode, thus it has access to all the traffic occurring on the subnet. Machine C is the management console for the firewall; it accepts directives from the SEM system

(Machine D) to disallow access for a set of IP addresses into the network. The cracker itself is assumed to be coming in from an external network.

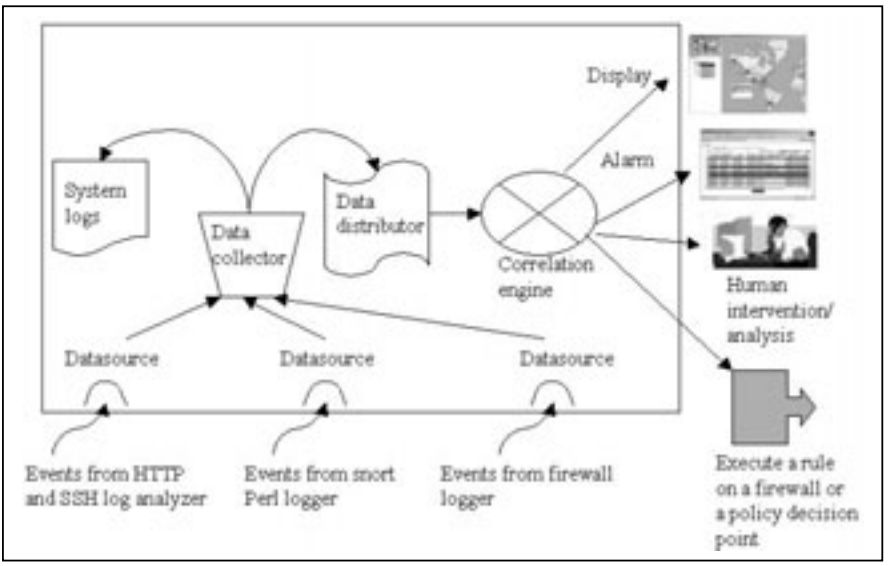


Figure 4: SEM Framework.

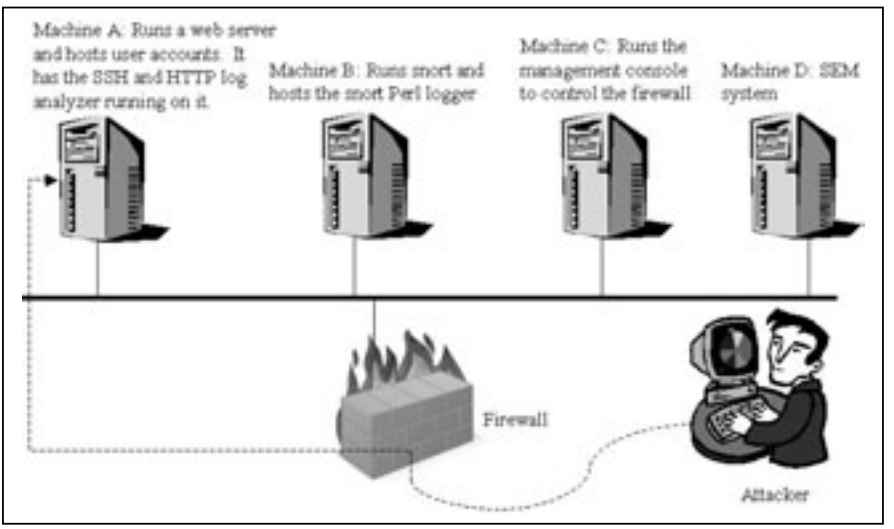


Figure 5: Laboratory Setup.

Typical security attacks follow a pattern: reconnaissance to find vulnerabilities, followed by a break-in, followed by expanding access to more system and network resources. This cycle is depicted in Figure 6(a). Mirroring this pattern, the attack proceeds in four stages. Note that in a deployed system, the attack can be stopped on the first occurrence of its detection. However, since we were interested in allowing the attack to proceed in a controlled environment, we at times permitted the attack to continue before having the firewall issue a block on the attacker's IP address.

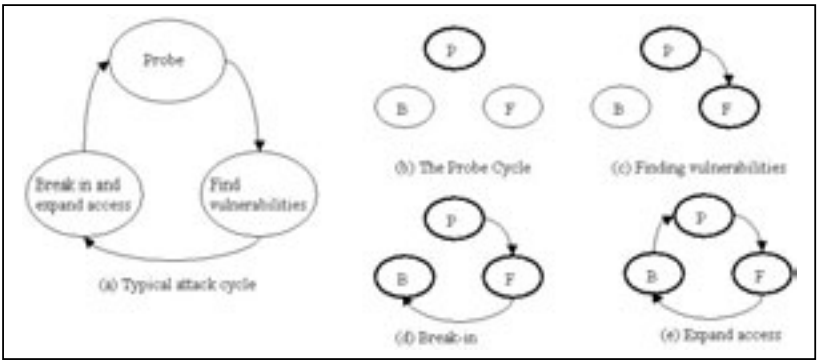


Figure 6: Attack cycle states.

Stage 1: The first stage is very preliminary; in the attack cycle of Figure 6, the attacker is in the probe cycle state, Figure 6(b). Here, the cracker attempts to reconnaissance the subnet using nmap (nmap, or „Network Mapper" is an open source tool for network exploration or security auditing; see http://insecure.org/nmap). When the attacker does this, many events are generated simultaneously: snort generates events about network scanning and sends them to the SEM system. In addition, the firewall also generates rate-limit violation alarms (i.e., incoming packet rate or session-establishment rate has exceeded a set threshold). All these events are sent to the SEM system, which has to correlate them and associate them with the same attack. Upon correlation, the SEM system issues a „Possible ICMP Probe" advisory. At this stage, we allow the attack to proceed into Stage 2.

Stage 2: Based on the output of the nmap probe, the attacker has now zeroed in on Machine A, the machine that hosts a web server . Using an open source web scanner called nikto (http://www.cirt.net/code/nikto.shtml), the attacker now tries to exploit a known vulnerability in the web server (in the attack cycle of Figure 6, the attacker is in the finding vulnerabilities state, Figure 6(c).) Nikto performs comprehensive tests against web servers for multiple items, including over 3300 potentially dangerous files/CGIs, versions on over 625 servers, and version specific problems on over 230 servers.
Since it is not designed as an overly stealthy tool, it's fingerprint is fairly obvious in log files. The HTTP log analyzer running on Machine A detects activity consistent with accessing well-known vulnerabilities; it sends a security event to the SEM system.

The SEM system correlates this event with those of Stage 1 and issues a denunciation to Machine C, which runs the management console to control the firewall. The denunciation results in the firewall blocking packets with the source address of the attacker. The attack has now stopped.

Stage 3: Undeterred, the cracker uses a new IP address to continue the attack. Having failed to exploit any vulnerabilities in the web server, the attacker now turns his attentions to the host itself. Using nmap, he performs a TCP and UDP port scan on Machine A. The snort instance on Machine B as well as the firewall itself generate security events; the snort instance generates a port scan alert event and the firewall generates a quality of service event. As in Stage 1, we allow this particular attack to continue and do not issue a block for the attacker's address.

Stage 4: Using the new IP address, the attacker performs a port scan of higher numbered ports. He finds out that a high-numbered TCP port (10022) is open, i.e., in listen mode. In many systems, users learning network programming or system administrators often keep servers running on a port. Many times, these servers are benign and don't do much damage if compromised; but at other times, the servers may inadvertently release information that will enable an attacker in mounting an effective attack. This was the case with the server on port 10022. When a connection is made to the server, it prints a listing of the home directory on the host. The attacker now has the login names of the users on that host. Using the login name, it is trivial to run a password cracker to successfully guess passwords of one or more user accounts. In the attack cycle of Figure 6, the attacker is now in the break-in state, Figure 6(d).

Armed with possible passwords, the attacker attempts to secure shell (SSH) into the system. Note that the inbound ssh connection is not detected by snort since is very well a normal occurrence in any network. (we could have configured the SSH log analyzer to issue a denunciation upon observing three failed login attempt for any user name to thwart such an attack; but in our scenario we did not do so and allowed the attack to proceed.) The attacker now has successfully logged into the system by guessing a password to an account by the name of „demo".

Since the reconnaissance and break-in have occurred, the attacker moves to the third state: attacking more systems (Figure 6(e).) To do this, the attacker uses the *wget* utility to download a malicious program from an external website at his disposal (since *wget* uses HTTP and most firewalls are configured to allow outbound HTTP connections to go through, this would not be tagged as an anomalous event.) The attacker retrieves the malicious program and tries to innocuously save it in the local directory. He then executes the program, which makes an outbound connection. As soon as an outbound connection was made to an arbitrary server, snort detects this and sends an unknown-outbound-connection event to the SEM system; the source port on machine A used for this connection is sent to the SEM system as well. The SEM system decides that this is an anomalous event and sends a directive to Machine A that causes it to perform a self-check (the directive includes the source port from which the connection emanates.) Using the *lsof* tool, a program on Machine A first figures out the owner of the source port; it subsequently runs the file integrity check on the home directory belonging to the

owner of that source port (account „demo“.) The integrity check uncovers the malicious program that the attacker had saved before and the SEM system is notified of this event.

At this point, human intervention is a must; the attacker has successfully breached the defenses and commandeered Machine A. The SEM system isolates Machine A by having the firewall drop all packets going to it or originating from it and notifies a human operator to intervene.

This scenario has demonstrated the need for a central SEM platform that can co-ordinate a response to a possible attack. Without such a platform, a network operator would have a coarse view of the events occurring in the network and may not be able to piece together the events that could prove to be a precursor to a larger attack.

## 5 Observations and Lessons Learned

There were several observations from constructing a SEM system that drive the lessons learned from this exercise.

**Network fault management complements but is different from SEM**

A fault management system and a SEM system are very similar in some of the key functions that they perform. They both receive, interpret, threshold, and correlate events from alarms from a wide range of devices. In fact, our SEM system was built upon an in-house high-performance fault manager containing an expressive correlation engine. Thus, it is interesting to ask whether a fault management system be transformed successfully in a SEM system. As a result of our work, we have reached a conclusion that while these systems perform similar operations on network events, security event management has some important needs that are not met by a network fault manager.

The primary function of the fault manager is to determine the underlying root cause of multiple received events, facilitate a remedy of the problem, and suppress the display of those events that represent symptoms of the root cause. A SEM system performs a similar function. An attacker may trigger the receipt of events of many different types from many sources and it is the function of the SEM system to determine the nature and source of the attack (the „root cause“) from these events, facilitate the mitigation of the attack, and suppress the reporting of those events that are symptoms or evidence of the attack.

However, while in fault management the root-cause fault might be intermittent and may even evolve into a larger problem with time, the root cause is generally static and the appropriate solution to address the fault remains valid once the root cause is found. Importantly, the root cause for the fault is normally a single component in a fixed geographical location. Furthermore, unless the root cause is a fire, flood, or other threat to the larger infrastructure, the damage caused by the event rarely escalates significantly with time. By contrast, a network attack generally proceeds in phases, often with

different attacking/probing sources and attacking/probing destinations as the attack evolves. The type and volume of events associated with the different phases of an evolving attack will change dramatically as the attack proceeds. The interpretation of the security events ("root-cause)" may proceed from "topology mapping," to "vulnerability probe," to "host vulnerability exploit," to "new bot detected," as the attack progresses. Consequently, there is an opportunity to detect the precursors to a full-fledged attack and an opportunity to mitigate the attack that has no common parallel in the fault management domain. If appropriate mitigation is not performed quickly, the damage can escalate dramatically with time.

**Network attacks are dynamic in nature and the symptoms can be difficult to detect and threshold**

There are great many possible categories of root causes for network problems, but a relatively small and predictable set of symptoms for identifying the root cause of a network fault. Consequently, a relatively static configuration for thresholding, event correlation and event suppression can diagnose these root causes effectively. The tools and interfaces for configuring a network fault manager reflect the relatively stable nature of the volume and type of the events expected. The threshold settings are relatively stable and generally require adjustment only as the network infrastructure or topology is modified. The correlation rules change infrequently and depend upon topology information in very simple ways. By comparison, network attacks and the precursor probes to these attacks come in fewer varieties, but the events that announce them can vary widely in type and in volume for different instances of an attack, and different phases of a single attack. A network probe can be brute-force and easily detectable or it can be carefully crafted to be difficult or impossible to detect. Consequently, it is impossible to create a set of event thresholds and correlation rules to reliably detect any but the hastiest and naive network scans and vulnerability probes.

Automatic or facilitated action is effective and valuable in addressing network attacks and is a valuable feature in a SEM system. A relatively small number of easily implemented solutions can address a wide range of network attacks. In contrast, because of the wide range of possible problems and the wide variety of network elements that can be the source of a root cause, it is not often feasible to mitigate or resolve network faults by automatic action, and automatic action is rare in all but the most specialized fault management devices.

Because the sources and varieties of network fault events are so open-ended and the collection of possible root-causes is essentially unbounded, a fault manager must provide the means for very flexible event interpretation and must be equally open-ended in the ability to define new kinds of alarm. To accommodate open-ended events and extensible alarm definitions, the mechanisms for event interpretation, thresholding, and correlation must be completely flexible and general. These requirements impose two important consequences for the internal architecture of these systems.

First, it is often necessary to create multiple alarms from a single event in order to apply multiple thresholding rules against different "views" of the event. Because the alarm

generation is so open-ended and thresholding and correlation mechanisms are necessarily very general, alarms are generated from events, and correlation alarms are generated from other alarms as arrays of name-value pairs in text form. As a result, as the alarm moves through the system, the arrays are repeatedly searched for field name matches, and the value fields are repeatedly parsed, and checked for correct syntax as they are communicated through the fault management system. Although this approach is very flexible and open-ended, it does not scale well as the volume of incoming events and the rate of thresholding and correlation operations increases.

Second, because the thresholding and correlation features of a fault manager are necessarily very general they are not always a good match to the capacity requirements of a SEM system. For example, the algorithms and data structures used within a fault management system for relating multiple failures to the failure of a shared network resource will probably not scale well when applied to the similar problem of detecting attacks originating from many sources or to many targets.

**Event thresholding, correlation and mitigation must be distributed and pushed down to the lowest possible level**

Because the volume of events received during an attack is generally much greater and less predictable than would normally be observed in a fault manager, without some care in the design of the detection architecture, the flood of events received during an attack can easily overwhelm the SEM system. Fortunately though, because security events are correlated into a smaller number of root causes, events can be correlated and summarized in a distributed manner forwarding a much smaller volume of event reporting to the central SEM system.

In a security event detection architecture, it is critically important to push event thresholding, correlation, and suppression down to the lowest possible level. For example, many firewall devices, including the one we used in our investigations provide features to set thresholds for traffic rate at the level of an individual session, a firewall rule, a particular network interface, or an entire rule set. In the firewall device we used, alerts can be generated for these events down to the level of rules but (for very good reasons) not to the level of individual sessions. In addition, the generation of logging records for session begin and termination can be enabled at the rule level. We found that if we were to naively enable these event reporting features and deliver the events directly to the SEM system, during a DoS attack or even during a benign traffic spike, the volume of information generated quickly overwhelmed the ability of the SEM system to digest it. Consequently, the logging features of the firewall device were typically employed very selectively and it is often not until the firewall reports that a resource threshold has been reached that the network operators are even aware that an attack has been underway. In a modern high-capacity firewall it can take hours to reach this condition.

Pushing thresholding and correlation features for DoS attacks down to the level of the firewall logging system would be a good, and necessary first step, but to address the problem effectively, we found that it would be necessary to implement new thresholding and correlation features within the firewall device itself. Furthermore, sampling of the event traffic and adaptive tracking of the top sources of firewall events would be necessary if we were to be able to perform this function effectively.

**Security event records must be designed for SEM system consumption**

When we applied firewall devices as security event detection points and forwarded the session events to the SEM system we discovered that the content and formatting of the event records was not well suited to interpretation by the system. Important session data was present in both the session start record and the session end record but there was no reliable identifier with which to associate the two records.

Furthermore, in the event that a session start record was discarded in the firewall by the automatic throttling mechanisms, important data would be lost. In our opinion, the best solution would be to provide the operator with the option of having the firewall device deliver all of the key fields in the session termination, and any intermediate event records. In addition, we decided that it was important to have every event record that was associated with a session contain an identifier to reliably associate those events. A similar example that we encountered was the way in which rules were identified in the session records with which they were associated. In the firewall device we used, the rule was named in the session record using the numerical order of the rule within the ruleset. The insertion or deletion of a rule would have the consequence of changing the identifier for the rule making it difficult to interpret the preceding session events for temporal ordering. In our opinion, having a reliable identifier to associate session records with the rules that enabled them is a key requirement for this class of devices.

**Remediation of security events**

Network operators whom we talked to during the early phases of constructing the SEM system recognize the value in having the security event management system propose and facilitate action to remediate security events. However, they have legitimate concerns about having such a system automatically generate and activate firewall policies and they have similar concerns about having the security event manager automatically block or terminate user sessions. Firewall policies often contain over 1000 hand-crafted rules and are modified infrequently and with great care. Automatic insertion and deletion of rules from these rule sets may be outside of the comfort level for many network operators. We have also observed reluctance to having the security management system automatically terminate sessions from traffic sources that have been judged to be sources of attack or egregious policy violation. Consequently, in addition to session termination and host blocking, we employed rate limiting on session establishment, threshold limits on session data rates, and redirection of sessions from suspect hosts to limit the damage that a suspect source can do. To address concerns about automatic policy generation, we employed modification of the host lists associated with source address matching on

certain rules to accomplish policy modification in a very constrained and well understood way. The redirection of sessions is accomplished using the policy-based routing features of the firewall. In our experimental network, we can redirect all or a subset of the sessions (e.g. HTTP/HTTPS) from the suspect sources through an intrusion detection system or into a sandbox server.

The firewall device that we have used in our work supports a device-independent zone-based security policy. With this design, a single collection of firewall policies are shared across an entire network by many firewall instances. This simplifies the remediation of attacks in a large network environment by enabling a single remediation action to be applied across the network avoiding the necessity to pinpoint the entry point of the attacker.

In summary, the lessons that we have learned from our work are:

1. Automatic action or operator-approved mechanized action for problem mitigation is a more feasible, valuable and desirable feature in a SEM system than it is in a fault manager.

2. Topology information is much more easily applied in a fault management system than it is in a SEM system.

3. Built-in correlation rules and analysis features for the detection and classification of security events is valuable in a SEM system but has no parallel in a fault manager.

4. A fault manager lacks the features necessary to efficiently correlate the events of a large scale DDoS attack.

5. Pushing down event correlation and suppression into detection devices (IDS), network element event logging systems, and network elements is crucial to effectively countering DoS and DDoS attacks in a large network.

6. Network elements that act as detection points may require modification of their event reporting mechanisms in order to work well as an element of a SEM system deployment.

7. The mechanisms provided for remediation of an attack must be designed with care to address the reluctance of the network operator to the introduction into the network of automatic policy control.

# 6 Agenda for Future Research in SEM Systems

The manner by which security events get to a SEM system today is largely ad-hoc; there isn't a formal language by which a SEM system can indicate interest in a subset of security events from the edge devices, nor is there a formal language for describing the events from the devices to the SEM system. While SEM systems may have a limited ability to reconfigure a device (allow or drop packets with certain IP addresses), they have no ability to query it pro-actively. Building SEM systems today amounts to performing integration work for the discrete pieces to communicate with each other. Continued research in SEM systems is essential to ensure that this is not the case in the future. In this section, we outline a research plan for SEM systems directed at solving the associated problems we have observed as a result of our work.

### Better Network Reconnaissance Techniques

Much of the focus of today's system is on the detection of denial of service attacks. While still important, interest is rapidly growing in attacker activities that can be characterized as low-level scanning for network reconnaissance. Such activities are performed by skilled hackers or botnets as a means of understanding the vulnerabilities of a target network prior to exploitation. These and other activities are largely undetected today and collection of correlation techniques will have to be created to account for them.

### Developing Resilient Protocols

Today, a SEM system is largely a one-directional system in that information is fed into the SEM. There is little in the way of two-way communications to, for instance, query or reconfigure detectors, or take action to reconfigure network devices to mitigate an attack. To do so, resilient protocols need to be developed for communication flow from the devices to the SEM system and from the SEM system to individual devices.

Designing a resilient protocols for communication is an open research problem in SEM systems. Ironically, it is precisely when a network is under attack that it may be least able to devote bandwidth resources for informing a SEM system. A protocol designed for communication from the discrete devices that are reporting their status to a SEM system thus needs to be resilient in the face of an adverse network. It should ensure that the information that is sent to the SEM is idempotent, self-contained and requiring minimal overhead in form of retransmissions and acknowledgements. For example, if five copies of a packet are sent by a device to a SEM system to increase the probability that at least one arrives at the SEM system, then even if the packet loss is 20% (and drop rates in today's network are typically under 5%) the odds are extremely high that at least one copy of the packet will get through. With a 20% packet loss and a five copies transmitted, assuming that the packet transmissions are spread out so that all losses are independent the odds are still greater than 99.6% that at least one copy gets through.

In the same vein as above, a resilient and universal protocol is required for the SEM system to impose policies and control the discrete devices in a network. Unlike the protocol discussed in the above paragraph, this protocol flows from the SEM system to the devices, thus it needs support in the individual devices themselves. Today, the commercial SEM system vendors depend on the device manufacturer to drive this communication. Thus, if the device can only be controlled using a proprietary protocol over TCP, then the SEM system would open up a TCP connection to the device and send it appropriate directives to cause it to change its behavior. As noted previously, a network under attack may not be quick to establish a TCP connection and to transmit directives over this reliable connection. Instead, what may be required here is the same mix of idempotency and multiple packet transmission as we discuss above. Furthermore, the protocol may well need to be standardized if it is to be implemented in a wide variety of edge devices.

**Policy Languages and Rule-based Systems**

Today's solutions are largely concerned with the event management of security devices such as firewalls, intrusion detection systems or intrusion prevention systems. The reality is, however, that every network device is a collector of information that may lead to a better understanding of what kinds of attacks are progressing. This information, if collected today, is usually done by enabling syslogs (system logs) for recording purposes. However, there isn't a formal mechanism to specify what information should be collected and how. We believe that a formal, universally implemented and verifiable mechanism is essential to for identifying security attacks in early stages.

Another area of research is the need for a security policy language for SEM systems. A robust policy language should be expressive enough to describe events of interest and to perform causal analysis. Related to this is the exploring the possible use of rule-based languages in a correlation engine: information that arrives at a correlation engine is typically in the form of facts („File X is accessed by IP address Y," „File X has been modified," „Web server is being scanned for vulnerability analysis from IP address Y," etc.) Once these facts are input into the system, it would seem rule-based logic langauge (like Prolog or the CLIPS expert system, see http://www.ghg.net/clips/CLIPS.html) should be able to run through its predicates to derive necessary alarms and denunciations for the remediation of attacks.

**Device Modeling**

Device modeling is yet another important aspect for further study in SEM systems. To achieve automatic and adaptive control of security monitoring and control points, it will be necessary for the SEM system to have knowledge of the characteristics of each of the controlled devices and of the location of each device in the network. Ideally, the SEM system should be able to query the capabilities and the physical location in the network of each of its detection and control points to be able to discover the best resources at each point in the network for monitoring network events and mitigating attacks. In order to do this, the security devices employed by the SEM system must be described to it in a

common model and be available to the system in a knowledge base that can be queried to adapt the monitoring and attack mitigation to changes in device inventory and network topology. The design of a device model that can encompass the full range of detection and control capabilities of these devices and the creation of a knowledge-driven system that can adaptively apply these resources, collecting feedback on the effectiveness of these actions to automatically improve behavior is a rich area for further exploration.

**Correlation Rules and Network Topology**

Correlation rules and the desired actions to mitigate an attack are inseparable from network topology. Specifics about the network topology, IP addresses, domain names, routing policy, network architecture, and network services are embedded in both the rules and actions that are encoded in a SEM system. Any change to network topology, therefore, affects and probably breaks the ruleset. Further research is required in this area to study how network topology impacts correlation rules, and to perhaps even allow for an automatic adjustment to the ruleset when the topology changes.

**Integration with Operations, Administration, Maintenance and Provisioning**

Most of the time, the output of an SEM system will be a rule that is enforced in the traffic admission component to mitigate an attack. But in some cases, an attack may simply consist of one packet that ends up crippling key servers such as a Session Initiation Protocol server (SIP is an Internet telephony signaling and services protocol [14]). For example, a documented attack in SIP consists of sending a request to a forking proxy that does not perform loop detection. As a result, the proxy is forced to maintain $2^{69}$ transactions in memory; clearly an untenable task [16]. If a SEM system observes the same SIP request traversing the network multiple times, it can restart the SIP proxy so that the proxy does not send out the looped request all over again. To this extent, is may be worth investigating into the feasibility of coupling of a SEM system with the OAM&P framework on a host.

**Developing Human Computer Interaction for security (HCISec)**

While one goal of a SEM system is to minimize human involvement, human input is still required. This includes entering network configuration information into various components, including the policy manager, detectors, correlators and simulators, defining policies and responding to alerts in cases where responses have not been automated. The APIs for modeling the network within the components and defining policies must not be overly complex or abstruse in order to limit inaccuracies and errors in human entered data. The manner in which alerts and supporting information is presented to humans must be carefully considered. One common problem today when presenting alerts to users is the quantity of information, both in the number of alerts and supporting information. The ease of using the system and the inclusion of safeguards against human error must also be considered. Approaches have typically followed that taken in network management where there is generally one root cause alarm and many

secondary and tertiary alarms. Processing has evolved to suppressing secondary and tertiary alarms, and displaying the root alarm to the user. While this approach has been suitable for network management because alarm hierarchy is fairly well understood at both the network and component levels, this is not appropriate, in general, for SEM systems where security events do not adhere to a known hierarchy and do not necessarily follow known sequences of events.

## Bibliography


[1] Bacon, J.; Moody, K.; Bates, J.; Chaoying, M.; McNeil, A.; Seidel, O.; Spiteri, M.: Generic Support for Distributed Applications, IEEE Computer, 33(3), 2000, pp. 68-76.

[2] Carzaniga, A,; Rosenblum, D.; Wolf, A.: Design and Evaluation of a Wide Area Event Notification Service, ACM Transactions on Computer Systems, 19(3), 2001, pp. 332-383.

[3] Cugola, G.; Jacobsen, H-A.: Using Publish/Subscribe Middleware for Mobile Systems, ACM Mobile Computing and Communications Review, 6(4), 2002, pp. 25-33.

[4] Cugola, G.; Di Nitto, E.; Fugetta, A.: The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS, IEEE Transaction on Software Engineering, 27(9), 2001, pp. 827-850.

[5] Debar, H.; Curry, D.; Feinstein, B.: The Intrusion Detection Message Exchange Protocol (IDMEF), IETF RFC 4765, March 2007. Available online at http://tools.ietf.org/rfc4765..

[6] Devitt, A.; Duffin, J.; Moloney, R.: Topographical Proximity for Mining Network Alarm Data, Proc. of ACM SIGCOMM Workshop on Mining Network Data, Philadelphia, USA, 2005; pp. 179-184.

[7] Duan, Q.; Hu, C.; Wei, H-C.: Enhancing Network Intrusion Detection Systems with Interval Methods, Proc. of ACM Symposium on Applied Computing (SAC), New Mexico, USA, 2005; pp 1444-1448.

[8] Ertoz, L.; Eilerston, E.; Lazarevic, A.; Tan, P-N.; Kumar, V.; Srivastava, J.; Dokas, P.: MINDS – Minnesota Intrusion Detection System, Next Generation Data Mining, MIT Press, 2004.

[9] Feinsten, B.; Matthews, G.: The Intrusion Detection Exchange Protocol (IDXP), IETF RFC 4767, March 2007. Available online at http://tools.ietf.org/rfc4767.

[10] Julisch, K.: Clustering Intrusion Detection Alarms to Support Root Cause Analysis, ACM Transactions on Information and System Security, 6(4), 2003, pp. 443-471.

[11] Liu, L.; Li, Z.; Xu, L.; Chen, H.: A Security Event Management Framework Using Wavelet and Data-Mining Technique, Proc. of IEEE International Conference on Communications, Circuits and Systems, China, 2006; pp. 1566-1569.

[12] McQuaid, R.; Security Information Management for Enclave Networks (SIMEN), Mitre Corporation, unpublished. Available online at http://www.mitre.org/news/events/tech07/2606.pdf, 2007.

[13] Ning, P.; Cui, Y.; Reeves, D.; Xu, D.: Techniques and Tools for Analyzing Intrusion Alerts, ACM Transactions on Information and System Security, 7(2), 2004, pp. 274-318.

[14] Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M.; Schooler, E.: SIP Session Initiation Protocol, IETF RFC 3261, June 2002. Available online at http://tools.ietf.org/html/rfc3261.

[15] Sekar, R.; Gupta, A.; Frullo, J.; Shanbhag, T.; Tiwari, A.; Yang, H.; Zhou, S.: Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions, Proc. of ACM Computer and Communications Security (CCS), Washington, DC, 2002, pp. 265-274.

[16] Sparks, R.; Lawrence, S.; Hawrylyshen, A.: Addressing an Amplification Vulnerability in Session Initiation Protocol (SIP) Proxies, Internet-Draft, work in progress, March 2007. Available online at http://tools.ietf.org/html/draft-ietf-sip-fork-loop-fix-05.