ANATOMY OF A SECURE AND SCALABLE MULTIAGENT SYSTEM FOR EVENT CAPTURE AND CORRELATION

Timothy Nix, Kenneth Fritzsche, Fernando Maymi
Department of Electrical Engineering and Computer Science, U.S. Military Academy,West Point, New York

Keywords: Software Architectures, Component-Based Systems, Component Composition, Multi-Agent Systems, Event Correlation, Distributed Intrusion Detection.

Abstract: Event monitoring and correlation across a large network is inherently difficult given limitations in processing with regards to the huge quantity of generated data. Multiple agent systems allow local processing of events, with certain events or aggregate statistics being reported to centralized data stores for further processing and correlation by other agents. This paper presents a framework for a secure and scalable multiagent system for distributed event capture and correlation. We will look at what requirements are necessary to implement a generic multiagent system from the abstract view of the framework itself. We will propose an architecture that meets these requirements. Then, we provide some possible applications of the multiagent network within the described framework.

1 INTRODUCTION

Correlating multiple sensor readings in real-time or near-real-time is process intensive when the sensors generate data at even a moderate rate. In our experience, it is not uncommon for servers in large organizations to generate over two gigabytes of event data per day. Centralized correlation with any but the simplest rules would be severely taxing to process in near-real-time. However there is a solution. Agent architectures are appropriate for problems that require systems to meet a variety of goals in a dynamic, unpredictable environment (Maes, 1992) and multiagent systems provide a distributed framework for problem solving that is too large for a centralized agent to solve because of resource limitations or the risk of having a processing bottleneck or single point of failure (Sycara, 1998). The definition and advantages of autonomous agents are described in detail in AAFID (Balasubramaniyan, 1998). This paper will examine some of the requirements that are necessary to a generic, hierarchical multiagent network designed to capture events and perform event correlation, present some architectural considerations that meet those requirements, and discuss some potential applications for the architecture.

2 REQUIREMENTS

In a multiagent system, the correlation workload can be distributed so that only events or statistics of significance are reported up the agent hierarchy. Obviously, this system can only be successful if the definition of interesting events can adapt to the ever-changing environment of production networks. However, when using a multiagent system to distribute the correlation workload, we want the architecture to preserve the security and integrity of the entire system, and make use of the existing sensors whose data we want to correlate.

In creating such a distributed multiagent architecture, it is necessary establish a set of requirements and constraints upon our system. These are much like Silva et al's "agent properties" which they refer to as non-functional requirements (Silva, 2003). They are desired system qualities that are not necessarily required for agents to complete their goals. These include requirements such as: (1) the system must be scalable; (2) it must be relatively simple to implement, configure and maintain; (3) it must be secure, meaning that the traffic generated by the correlation network can not be sniffed of spoofed; (4) it must not be limited by the network topology; (5) it must be self-healing; and (6) it must be time-synchronized.

- Scalability

In terms of scalability, we want to be able to grow our agent network as large as functionally necessary regardless of the size of the network. Depending on the application of the correlation network and the established correlation rule set, we could have as few as one agent for the entire sensor network or as many as one agent per sensor ($1 \leq x \leq n$: x is the number of deployed agents and n is the number of individual sensors on the network).

- Simplicity

In order to keep the agent deployment and distribution complexity to a minimum, we want all agents to be identical. In other words, the coded representation of each agent is the same. However, this does not mean that they all behave the same. Each agent's role and functionality is determined not by its binary code, but by data in its configuration file and the specific sensors it monitors. Configuration modification can either be performed on a host local to the agent or via a graphical user or web based interface that communicates strictly with the "uber-agent". The configuration changes are then disseminated to the necessary agents in the logical agent hierarchy.

We also want to avoid the need to develop new sensors or to modify existing ones so that they integrate into our framework. Modifications to sensors such as syslog or SNORT would seriously complicate deployment and could undermine security. To this end, we want the framework to tap into the data stores of a large variety of widely used sensors such as Microsoft® Windows™ auditing, syslog, SNORT, as well as others found in such devices as firewalls and Honeynets.

- Security

To enhance security and limit the traffic generated by the correlation network, each agent is configured so that it can only communicate with a limited number of other agents. This prevents rogue agents from being launched in order to corrupt or control the correlation network.

Each agent should contain a message management class which handles encryption and decryption of message traffic and key management. The message management class of one agent uses asymmetric encryption to communicate with the message management class of other agents. Using asymmetric encryption allows for both encrypted and digitally signed traffic. Encrypted traffic prevents sniffing and signed traffic prevents spoofing.

- Independent of physical network topology

The topology of the multiagent network is independent of the physical network. Thus, the employment of agents and sensors may or may not match the topology of the physical network. The only requirement is that agents must be able to communicate. Agents that are adjacent on the correlation network may be located on entirely different subnets. For example, the correlation network can be organized by sensor type with the sensors being distributed one per subnet or logical work unit. On the other hand, an agent hierarchy could be built to handle all web servers in the enterprise, regardless of where they are in the physical network. This requires agent hierarchies to be deployed across functional business units within an organization. For this reason, we do not want to restrict our multiagent network by constraints that may not apply to the application.

- Self Healing

The distributed nature of the system and the requirement that each agent's code be identical means that any agent can perform in the role of another agent. Thus, there is built in redundancy and we do not have a single point of failure within the system. This allows our agent network to have a method of recovering from the failure of a single agent, i.e. the network is self-healing. Any agent can assume the role of another agent in the logical agent hierarchy, and the agent network thus becomes resilient against attack or localized network failure.

The agent network needs to be able to recover from the loss of one or multiple agents within the hierarchy. It needs to be able to self-heal in the event it loses communication with other agents in the network. Within its configuration file, the agent could contain the address of one or more generations of its ancestry and progeny. Thus if communication is lost in either direction, communication can be re-established. In this manner, one (or multiple) failed agent(s) does not result in the failure of an entire subnet within the multiagent hierarchy.

- Time synchronization

The distributed design of the agent network requires a system be implemented for network time synchronization. This is particularly important when performing correlation of events across distributed sensors/hosts to resolve relative order of events and determine causality between events (Karp, 2003).

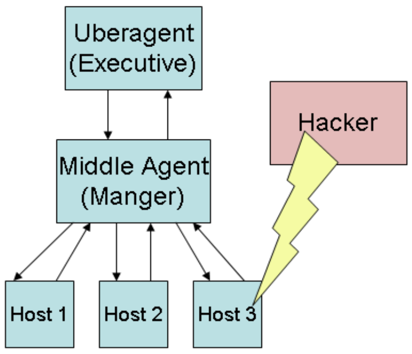

Figure 1: Agent Roles and Hierarchy. The above illustrates a malicious act (as defined in the agent's rule set) on Host 3. This in turn triggers the agent to send a report to the next higher agent in the hierarchy

3 ARCHITECTURE CONSIDERATIONS

With these requirements in mind, we next strive to develop an architecture that meets these requirements while maintaining a degree of independence from the actual application for which it will be used. Our architecture consists of the agents, which consult (correlate) sensor data. They do this by querying the sensors' data repositories and the sensors themselves, which provide the data tracking vital dimensions of the problem space. An agent consists of a communication package that provides secure communications with other agents; a query engine that can formulate database queries and search XML or flat files for specific information (still field based); and a correlation engine which processes query results and generates alerts and/or query replies to/from another agent. The agents are organized hierarchically with one agent at each tree or subtree root designated as the "uber-agent" as shown in Figure 1. A user interface ties into the "uber-agent" for managing the multiagent network.

• Sensors

In the context of this architecture, a sensor is anything that logs updates of a changing state either internal or external to the network. In the classical context, a sensor may measure or record physical phenomena such as temperature, air pressure, light, etc. But, a sensor could also "measure" a given dimension of the network, by tracking packets or logging events. Sensors can be imbedded in the operating system as proposed by Kerschbaum (Kerschbaum, 2000), or separate processes. Examples of such sensor measurements or recordings include log files produced by syslog, a firewall log, an alert log generated by an intrusion detection system, Windows™ events, etc. A sensor must satisfy two constraints: (1) it must measure a necessary dimension of our problem space; (2) and it must record its measurement in some sort of datastore. The datastore is not required to be on the same host as the sensor. We have thus far restricted a datastore to one of three formats (though others are possible): a relational database; an XML file; or a text file.

• Agents

An agent is a self-contained process that has the ability to perform queries on a datastore, correlate the results of the queries in accordance with the agents query rules, and communicate with other agents. Communication takes the form of queries, query results, or configuration changes. Communication is secure, meaning that the traffic can not be sniffed of spoofed.

The agent network is hierarchical in nature. At the top, a graphical or web-based interface communicates with the "uber-agent". The "uber-agent" communicates with several "sub-agents" and/or sensors. This can be seen in Figure 1. These "sub-agents" communicate with one or more "sub-agents" and/or sensors, recursively. Dispersion of agents matches the distribution of the datastores across the network. An agent is co-located on the same host as a datastore. One agent suffices to query all datastores located on a single host.

• Correlation Engine

The correlation engine is part of an agent. Within the context of our architecture, the correlation engine consists of two parts: (1) data correlation; and (2) alert generation. Data correlation stems from simple and compound Boolean and relational operations on data queries. A compound Boolean expression, or signature, can be easily parsed and transformed into one or more queries of the datastore. At this level, data correlation is a matter of pattern matching the data within a datastore to a signature (i.e. misuse detection).

Should a signature be matched then that may be enough to generate an alert. In fact, this is how many intrusion detection systems work. The problem with this approach is that it is limited to matching known signatures. We also need a way to recognize hidden or new correlations for which we have no signatures. One way of doing this is through anomaly detection based on a statistical analysis of "typical" network data. We register "normal" data over time and generate an alert when the correlation of data from one or more sensors flag "abnormal" behaviour (i.e. anomaly detection).

The architecture presented in this paper really is a framework which is independent of "how" the correlation is done. All agents can correlate their data in an identical manner, or agents can be specialized as in (Chatzigiannakis, 2004). Our architecture is suitable for use as a research platform for exploring distributed AI. Furthermore, the modular design of our framework makes it possible for various agents to correlate events differently and still maintain their sociability.

- Configuration

A configuration file for a given agent provides the agent with the information it needs in order to understand its environment. The configuration file will contain information regarding one or more levels of the agent's "parent" and "children" depending on the level of self-healing desired by the agent network. Additionally, information is also contained in the configuration concerning the sensor datastore(s) for which the agent is responsible. The agent's configuration is also established by a set of correlation rules that it applies locally on its own datastores.

4 APPLICATIONS

Our multiagent architecture is generic enough to be the basis for implementation in multiple applications. An example of a possible deployment of the multiagent network is shown in Figure 2 in which we see sensors distributed throughout a network. Each sensor logs its data either locally on the same host or remotely to a different host. Sensor logs take the form of a relational database, an XML file, or a text file. Agents are distributed throughout the system as well. To minimize the potential of queries being attacked, agents are collocated with the sensor logs. Agents query sensor logs and correlate the results. Based on the correlation rules, the agents could disseminate the results of its correlation to other agents for further action. Agents can have access to a special data store known as the central logging facility (CLF). The CLF can store either results of reports generated by agents, or raw sensor data used for further correlation. The intent is for storage of query results pending further action, correlation rules, configuration rules, etc. The multiagent network is hierarchical in structure and managed at the root by a user interface.

The architecture discussed in this paper was primarily developed for correlation of intrusion detection systems with system and network logs. The abstract and modular structure of the architecture makes it easy to implement for correlating any sensor network. In this or similar contexts, we could use the system to implement a system monitor or an intrusion detection system. We could also use the system to guide forensics of a network attack either as part of the network or as part of a honeynet. Or, we could use the same architecture in an entirely different way for environmental monitoring.

- System monitor

A system monitor could perform data mining and correlation on any logging process that monitors the state of a host, network, etc. In this context, sensors take the form of system logs such as syslog, httpd log, firewall log, router log, etc. Agents then perform distributed queries on the datastores (logs) of these sensors. Correlation rules are developed in order to match specific contextual criteria. For example, a rule could be developed to return data associated with events within a given time frame, or generate an alert if a certain number of failed log-ins, power interruptions, or other event of significance occurs within a specified time. From these rules and queries, the overall status of a network, functional systems (like all web servers within an enterprise) or workstations within a business unit can be determined and reported.

- Intrusion detection

One of the problems with many available intrusion detection systems is determining the proper balance between rules that produce an excessive number of false positives or potentially allow a false negative. In the first case, the network administrator wastes time chasing after ghosts, and in the second, a wily hacker remains undetected while having free reign of the network.

Using multiple individual intrusion detection systems as sensors (such as Snort, Border Guard by StillSecure, Cisco IDS, etc), forming a distributed IDS or dIDS (Einwechter, 2001), the multiagent network could potentially correlate the alerts generated by the multiple IDS with other system logs. For example, the IDS would log alerts to a file or database. An agent would query an associated IDS log and correlate the results with the results of other agents and their queries on other IDS or system logs in order to reduce the number of false positives generated. Additionally, the multiagent network could look for variations of attempted attacks by taking a rule which generated an alert and querying the system for data regarding things like all traffic from the attacking host, all traffic directed to the attacked port, etc. A different strategy might be to correlate the results of various statistical-based IDS with the results of other rule-based IDS. Then, the agent could develop new rules for the rule-based IDS based on the results.

Optionally, we could develop an IDS from scratch that is rule-based, statistical-based, neural network based, or based on any potential new research in intrusion detection. Using a packet sniffer that promiscuously logs all subnet traffic, the agent network could look for patterns in the data. For a rule-based approach, the agent's local query rules would be developed that match a potential attack signature. Queries on the sensor logs would be executed by the agents. Positive returns on the queries would generate an alert. Used in conjunction with perhaps a statistical correlation engine, or a domain specific AI, a sophisticated network IDS could be developed.

• Forensics

In the event that we detect a successful penetration of our network "after the fact", we will ideally need to pull our network offline and study how the hacker succeeded. In this scenario, computer forensics is a means to analyze the network events in order to determine what occurred during the attack. Good forensics will help us "generate" proper IDS rules to prevent hackers from using the same method of attack in the future. Our agent network, using the same sensor setup as used in intrusion detection or system monitoring can easily be used for forensics. Here, with an agent's configuration, correlation rules are generated at the interface in accordance with the demands of the security administrator. Thus, the administrator would select the constraints with which to filter network data. These constraints would be transformed into the agent network's query language and disseminated through the network as appropriate. Each agent would receive some form of the original query, parse the query, forward the query in part or in whole, and perform queries on its local datastores, as necessary. The results received from an agent's "children" are then consolidated with the results of the local queries and passed up to the agent's parent. The "uber-agent" passes the results to the interface which displays them for the administrator. Using this technique, a security administrator should be able to narrow down which network traffic applied to the hackers attack and glean from the logs an attack's modus operandi. Additionally, by exporting log data and reports off a compromised machine to the CLF on a recurring basis allows a redundancy that can be used to compensate for suspect logs from a hacked machine.

• Environmental monitoring

A completely different context for the same agent network would be one where the network sensors

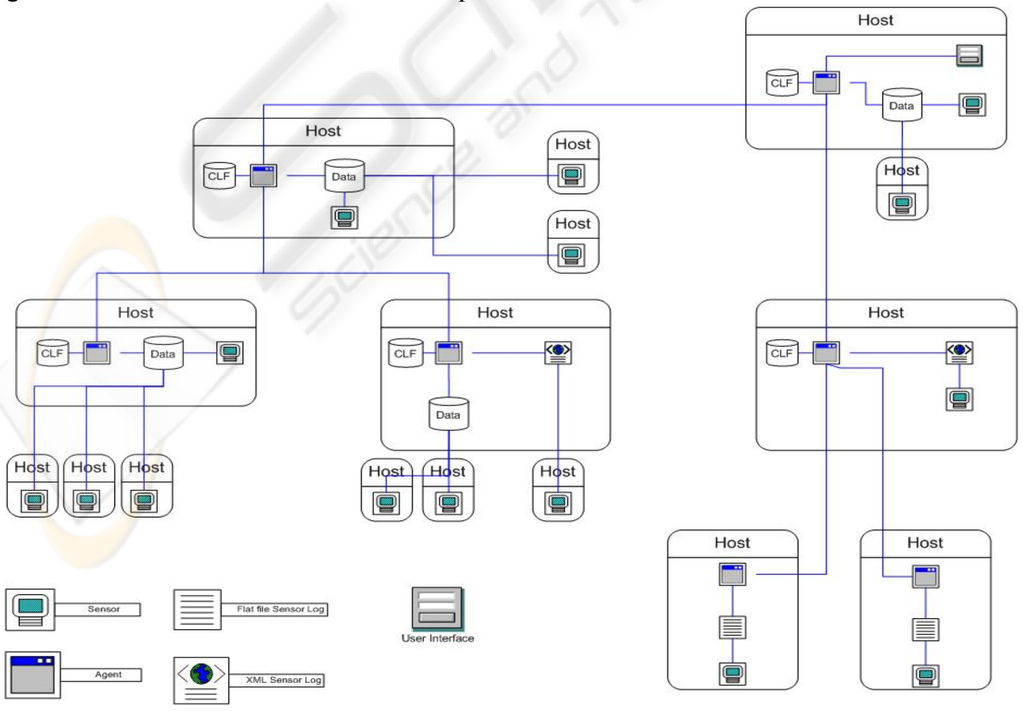

Figure 2: Picture depicting how sensors, agents and datastores can be deployed across a netwrok. Sensors must create logs, either on the local host or at a distributed log site. For each host with a log data store, there is a co-located agent that can query the data store and send information to agents higher in the hierarchy.

were actual physical sensors. Sensors such as those that measure power consumption, temperature, humidity, air pressure, wind speed, perhaps digital camera, motion detectors, or just about anything that digitally records a physical parameter can be used. Examples of correlating such data would be to generate an alert if the temperature dropped more than ten degrees in an hour. This type of system could be part of an IDS, a means to protect equipment sensitive to changes in weather/conditions, etc.

- Alert conditions (holistic vs. local)

By correlating sensor data across the entire network, we can potentially generate one alert condition for a local host or subnet and a different alert condition for the entire network. A local alert condition may be high if hacker penetration is eminent, but the services on the affected host are minimal enough that the host's compromise would not provide the hacker with much useful information. For example, an agent is monitoring a honeypot and an intrusion is detected. The result would be a low alert condition for the network. Also, we could change the rule set that an agent uses based on the alert level - additional parameters could be monitored, certain events could automatically be forwarded to the CLF of the "uber-agent", etc.

Another example, in an IDS application, if a subnet was recently port scanned, it might set its alert condition to "red" and use a looser set of rules that would otherwise have generated a high number of false positives. Other agents within the system could be notified of the threat and adjust their threat condition accordingly. The alert condition for high-value targets would be set to "red", some subnets might increase to "amber", and low-value systems might not change at all.

In an environmental monitoring system, perhaps the agents query sensor logs once every 15 minutes. If one of the sensors record a drastic change (temperature, air pressure, etc.) within one or more monitoring periods, the agents could be set to query the sensor logs once every minute.

5 CONCLUSIONS

A multiagent architecture shows great potential for solving problems in a distributed manner that a single agent could not process in a timely manner. Research in this area tends to focus on ways to implement a cooperative artificial intelligence. The architecture presented in this paper provides a means to separate the AI from the other important aspects of a multiagent system. The modular design allows for easy research, testing, and application of distributed AI systems in a variety of contexts. The scalability, simplicity, security, and robust nature of the architecture provide a common structure in which to compare and contrast competing paradigms for learning, cooperation, network timing, etc.

REFERENCES

Balasubramaniyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E., Zamboni, D. (1998, December). An Architecture for Intrusion Detection using Autonomous Agents. *Proceedings of the Fourteenth Annual Computer Security Applications Conference, pages 13-24*. IEEE Computer Society. Retrieved from http:// www.cse.buffalo.edu/~sbraynov/seminar%202004/papers/zamboni-agents1.pdf

Chatzigiannakis, V., Androulidakis, G., Grammatikou, M., Maglaris, B. (2004, June) A Distributed Intrusion Detection Prototype using Security Agents. *In 11th Workshop of the HPOVUA*.

Einwechter, N. (2001, January 8).An Introduction to Distributed Intrusion Detection Systems. Retrieved from http://online.securityfocus.com/infocus/1532.

Gopalakrishna, R., Spafford, E. (2001) A Framework for Distributed Intrusion Detection using Interest Driven Cooperating Agents. Purdue University. Retrieved from http://www.raid-symposium.org/raid2001/slides/ gopalakrishna_spafford_raid2001.pdf.

Karp, R., J. Elson, D. Estrin, and S. Shenker. (2003, April 11). Optimal and Global Time Synchronization in Sensornets. *Center for Embedded Networked Sensing Technical Report 0012*. Retrieved from http://www. eecs.harvard.edu/~mdw/course/cs263/fa03/papers/timesync-techrept03.pdf.

Kerschbaum, F., Spafford, E., Zamboni, D. (2000, November). Using embedded sensors for detecting network attacks. *Proceedings of the First ACM Workshop on Intrusion Detection Systems*. Retrieved from http://www.cerias.purdue.edu/homes/zamboni/ pubs/wids2000.pdf.

Maes, P. (1992) Modeling Adaptive Autonomous Agents. *Artificial Life Journal, Vol 1, No 1&2, pp 135-162*. MIT Press.

Silva, C., R. Pinto, J. Castro, and P. Tedesco. (2003, November 27-28). Requirements for Multi-Agent Systems. *Workshop em Engenharia de Requisitos, Piracicaba-SP, pp 198-212*.

Sycara, K. (1998) Multiagent Systems. *AI Magazine, Vol 19, No 2. pp. 78-92*. Retrieved from http://www-2.cs.cmu.edu/~softagents/papers/multiagentsystems.PDF.