

# ACTIVE MONITORING OF NETWORK SYSTEMS USING MOBILE AGENTS \*

ANAND TRIPATHI, TANVIR AHMED, SUMEDH PATHAK, ABHIJIT  
PATHAK, MEGAN CARNEY, MURALIDHAR KOKA, AND PAUL DOKAS

*Department of Computer Science  
University of Minnesota, Minneapolis MN 55455*

This paper presents the design and implementation of a mobile agent-based system for network monitoring. The design has been driven by the need for a system that can be dynamically modified, react to events as they occur, encode high-level administrative policies, support decentralized aggregation and correlation of data, and communicate and manage data securely. To address these needs the Ajanta mobile agent programming framework has been used to implement four types of agents: Monitor, Subscriber, Inspector and Auditor agents. These agents gather information to generate basic events and communicate with each other for system wide monitoring. Agents also correlate basic events to produce compound events that can represent high level policies.

## 1 Introduction

The goal of a network monitoring system is to ensure proper system operation by watching for inconsistencies in user activities, system configuration, and resource status. In an enterprise computing environment, one important functionality of a monitoring system is to guard system resources from misuse or intrusion. The system presented here is designed to support monitoring of different aspects of a computing environment, such as network traffic, host level activities, system configuration, file systems, and user activities.

Several factors make effective system monitoring difficult. Due to a large number of resources to be managed, a huge amount of monitoring information is produced without suitable tools to digest and correlate it efficiently. The heterogeneity of the infrastructures and the need to monitor different data sources increases the complexity further. This usually results in a significant delay in response to critical events by system administrators. In <sup>1</sup>, similar challenges are presented in the context of intrusion detection systems (IDSs). Monitoring systems need to inter-operate and correlate data across infrastructures with diverse technologies and policies. They need to cope with ever changing attackers' goals and their increasing abilities supported by sophisticated attack tools. Moreover, these systems need to provide facilities

---

\*THIS WORK WAS SUPPORTED BY NATIONAL SCIENCE FOUNDATION GRANTS ANI 0087514 AND EIA 9818338.

that easily integrate human analysis as part of event diagnosis. Monitoring systems also need to secure themselves against attacks.

In <sup>2</sup>, the mobile agent paradigm has been identified as a natural solution to implement monitoring systems. Regarding potential security solutions, William Wulf remarked <sup>3</sup>, "...Instead of having perimeter defense, you have lots of agents running around seeing if something bad is happening and attacking when it does". In mobile agent-based monitoring, autonomous software agents migrate to remote hosts and cooperate among themselves for system wide monitoring. Our agent based architecture is inspired by the paradigm of *cooperative security managers* <sup>4</sup>. We call it *active monitoring* <sup>a</sup>, since agents can alter their monitoring behavior based on critical events, and new monitoring functions can be added dynamically at runtime. Mobile agent based active monitoring utilizes mobile code technology, and incorporates the capabilities of software agents, namely autonomy, dynamic adaptability, and their ability to react and cooperate based on goals. AAFID (Autonomous Agents For Intrusion Detection) <sup>6</sup> is one of the earliest systems to propose agent mobility in IDSs. However, only a few monitoring systems <sup>7,8,9,10</sup> have utilized mobile agents. These agent based systems use a small number of agents and in many cases are primarily motivated by the need to reduce network traffic. These systems do not fully exploit the active monitoring paradigm, where agents can cooperatively adapt to changes in the monitoring environment.

We have identified several paradigms for mobile agent based network monitoring in <sup>11</sup>. Here, we present a monitoring system implemented using Ajanta mobile agent programming facilities <sup>12</sup>. The following goals have driven the design of this system.

**Dynamic extensibility and configurability:** Monitoring functionalities of an environment can change due to hardware or software reconfiguration, change in administrative policies, and addition of new monitoring tools. Moreover, it should be possible to enhance the monitoring system's capabilities in response to new attack scenarios.

**Active monitoring:** A monitoring system needs to alter its detection policies in response to critical events. For example, in case a critical event is detected, the monitoring system may start operating at an increased level of alertness and initiate monitoring of some additional events.

**Decentralized data aggregation and correlation:** The system should

---

<sup>a</sup> Active monitoring is different from the active programmable network (APN) <sup>5</sup> paradigm, which uses mobile code as part of network packets for altering the functionalities of switches/routers.

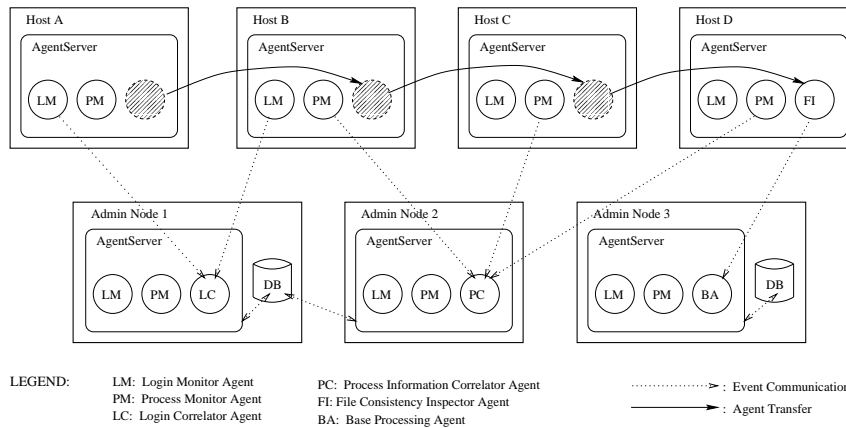


Figure 1. An agent-based framework for Network Monitoring

be able to support any desired architecture for decentralized correlation of data, generated by diverse and distributed sources. This requirement is motivated by the need for scalability and support for multiple administrative domains.

**Policy-based monitoring:** The system administrators need to be able to specify monitoring goals using high level policies. These policies can be specific situations to be monitored, e.g. no user should have access to more than two accounts, or actions that need to be performed when such situations occur.

**Security of the monitoring infrastructure:** The monitoring infrastructure itself needs to be secure from attacks. Also, the data gathered by the monitoring system needs to be communicated and managed securely.

**System performance:** The monitoring system should not interfere with the normal functioning of the monitored hosts. The system should not be resource intensive in terms of CPU cycles or memory used.

The main contributions of this paper are the design and implementation of a mobile agent based monitoring system addressing the above design goals, and our experiences with it. Section 2 presents the system design and architecture. The implementation and our experiences with the system are presented in section 3. Related work and conclusions are presented in sections 4 and 5 respectively.

## 2 Overview of the System Design and Architecture

Ajanta<sup>12</sup> is a secure Java-based framework for distributed programming with mobile agents. In Ajanta, mobile agents encapsulate code and execution context along with data, and they are executed on behalf of a user, termed *owner*. The Ajanta system provides *agent servers* to host mobile agents, a set of primitives for the creation and management of agents, and a global naming service.

Figure 1 shows a typical system organization in our agent based monitoring framework. Each node in the monitored environment executes an *agent server* which allows migration and installation of new agents for event monitoring, data collection, and correlation. This system uses four types of agents: *Monitor*, *Subscriber*, *Inspector* and *Auditor Agents*. *System Management Stations*, which typically run on physically protected hosts, can launch *Monitor* agents equipped with several *detectors* containing logic for event detection. *Subscriber* agents can subscribe for certain events, and any desired subscription policy can be implemented. The *Inspector* agents can move around the system based on their itineraries, looking for suspicious activities, and gathering information for further processing.

### 2.1 Event Model

A *basic event* represents some significant change in the state of a resource to be monitored. Higher level *combined events* are derived from basic events by applying certain inference rules. We need a canonical definition and representation of events, independent of any operating system specific details. For example, the concept of events such as *login-failure*, *remote-connection-request*, or *disk-system-full* are present in most operating systems. The event hierarchy is presented in Figure 2. The *SyslogEvent* class represents the events generated from the system log files. The *ConnectEvent* class represents various kinds of connections to a host. Its subclasses are events corresponding to SSH, FTP etc.

### 2.2 Monitor Agent

In this system, *Monitor* agents perform functions of event monitoring and correlation of event data. Each *Monitor* agent carries with it several *detectors*. New detectors can be added to a remote monitor agent, and existing ones can be removed. This allows us to extend the monitoring capabilities of an agent. Associated with each detector is the definition of a *handler* object. Every agent implements the *RemoteControlInterface* defined by Ajanta, for

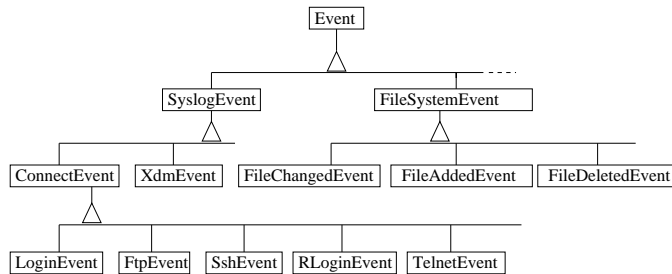


Figure 2. Event class hierarchy (partial)

its owner to remotely terminate or recall it. The agent also implements the *MonitorInterface*, which defines methods that allow another authorized agent to add, delete, suspend, or resume a subscriber for an event. Event detection may involve monitoring of log files or checking the utilization of system resources such as network ports or disk space. Detectors can also be triggered by events generated by other detectors. For example, the occurrence of a login event may trigger further detection to determine whether it came from an outside domain. Such information is maintained in the *trigger table*. Detectors triggered by certain events can thus be used for correlation and detection of compound events.

The handler object performs filtering and notification of the events to the subscribers. It can also store the events either in local or shared databases. In this system all shared databases are SQL based, whereas local databases can either be SQL or Prolog based.

### 2.3 Subscriber Agent

A *Subscriber* agent is extended from a *Monitor* agent by implementing the *SubscriberInterface*. The *SubscriberInterface* allows for an agent to receive events and thus act like a subscriber. A *Subscriber* agent can also add (delete) itself to a *Monitor* agent's subscription list, or suspend (resume) its subscription thus allowing a flexible system for event dissemination and filtering.

### 2.4 Inspector Agent

The primary role of an *Inspector* agent is to perform certain consistency checks. The *Inspector* agent works with a peer called a base agent which resides on a management station. The *Inspector* agent periodically visits hosts

specified in its itinerary, gathers the information and sends it to the base agent before migrating to the next host. This way it does not keep accumulating the data and remains lightweight. The base agent is responsible for processing the received data, generating events and communicating with the database.

In the initialization phase, an *Inspector* agent is used to construct the reference database. In the subsequent inspection rounds, this reference data is used to detect anomalies. We have chosen this specific architecture to keep the load on the monitored host as low as possible. However, the system architecture can support other models such as moving information processing to the monitored node.

### 2.5 Auditor Agent

In response to some critical events, an *Auditor* agent is launched to visit certain hosts to check and collect information for corroborating events that may possibly indicate an intrusion or abnormal situation. The *Auditor* agent's itinerary is dynamic and can be determined based on gathered data. These agents can also be created and launched to perform ad hoc monitoring tasks.

### 2.6 Agent System Security

Security is an integral part of the Ajanta design. It provides components for authentication, public key maintenance, access control, host resource protection, and cryptographic services<sup>13,12</sup>. Ajanta's security facilities are presented below:

- Access control on the acceptance of visiting agents is imposed by the agent servers based on the owner's identity.
- In Ajanta, each agent is given a set of unforgeable credentials, which contain the agent's name, its owner's name, and a set of privileges granted by the owner. Based on an agent's credentials, an agent server grants to the visiting agent restricted access to its local resources, such as privileges to read system log files.
- In Ajanta, agents at two different nodes can communicate with each other using RMI, provided their hosting servers permit them to open TCP-connections. Inter-agent communication can be authenticated.
- An administrator can also securely control its remote agents by either terminating their execution or recalling them. The secure *RemoteControl Interface* ensures that only the valid owner can disable its agents.

- Agents can also impose access control on its methods based on its own policy. A monitor agent can impose restriction on its operations such as adding/deleting detectors, handlers and subscribers. A subscriber agent can impose access control on event notification based on publishers' identities.
- In Ajanta, agent servers provide distinct protection domains for its visiting agents, which ensures that visiting agents do not interfere with each other.
- Ajanta's secure class loader ensures that the agent classes that are not core JVM system classes are loaded from a secure repository as specified by the agent's creator.

### 3 Monitoring System Implementation and Experiences

Here we present how the various paradigms presented in the previous section are used in building a monitoring infrastructure. This infrastructure integrates different agent based monitoring components to realize a comprehensive monitoring system.

#### 3.1 Syslog Monitor

The *Syslog Monitor* agents are used for monitoring system log files. Typically these files contain information related to user login activities, remote connections, and requests to system services such as SSH, FTP, etc. The core component of this agent is the *SyslogEventDetector*, which generates a basic *SyslogEvent*. This event triggers other detectors related to specific classes of events such as *ConnectEventDetector* and *XdmEventDetector*, which in turn perform specific pattern matching functions related to the event to be detected.

#### 3.2 Process Monitor

The *Process Monitor* agents are used for monitoring events such as existence of illegal processes, runaway processes, execution of unauthorized programs, or to check for a specific user's processes among others. It can detect process creation and termination events. The agents monitor process information such as owner, memory and CPU usage etc. The criteria the administrator wants to check about a process is specified through a policy file.

### 3.3 *File Consistency Inspector*

The *File Consistency Inspector* implements the file integrity checker functionality for our monitoring system. Files can be monitored for changes in various attributes such as file size, creation time, modification time, access time, inode, blocks allocated for storage, owner, and permissions. Along with the file attributes, the hash of the file contents is also computed using the SHA algorithm. This information is stored in the database and used as reference for later comparisons. *File Consistency Inspector* is based on the inspector agent paradigm. During each round of the agent, information about the monitored files is processed, and in case of a discrepancy events such as *FileAdded*, *FileDeleted* and *FileModified* are generated. The monitoring policies specify the hosts to be monitored along with the corresponding files and directories. The syntax for specifying these policies is similar to that used by Tripwire<sup>14</sup>. The administrator can program the agents to monitor the files at various alertness levels and at different frequencies.

### 3.4 *Host Fingerprinting Inspector*

The *Fingerprinting* agent is designed to monitor features of a host that should not change often. Similar to the *File Consistency Inspector* agent, the *Fingerprinting* agent uses the inspector agent paradigm. A fingerprint consists of a host's routing table, ports listening within a range specified by the system administrator, network interface configuration, operating system version, and other such details.

The motivation for using *Fingerprinting* agents is twofold. First, in a large network, failures of software components are often silent, even if these failures indicate significant events in terms of misuse or attack. For instance, if the SSH daemon on a host terminates, this will not be detected until a user attempts to use this service. The second motivation is that many types of attacks will modify this fingerprint. Often, intruders will start unauthorized services on a compromised host. The routing table of a host might be modified by an attacker to send packets to another host to monitor network traffic.

### 3.5 *Experiments in Active Monitoring*

We are currently running this system to monitor our lab environment, consisting of ten nodes, to conduct experiments. The *system management station* sends monitor agents to each host to be monitored based on a configuration file. This file contains information about the events to be monitored at each host. Our experiments make use of and integrate the above mentioned compo-

nents for comprehensive monitoring. Based on the events generated by these components, compound event detectors are implemented. Examples of such detectors based on specific scenarios are described below:

- Monitor the system for various types of *scans*, such as *ftp scan* and request for other unauthorized services, initiated from hosts both inside and outside the domain.
- Monitor for user login activity violating administrative policies, e.g., in our domain, a user is not allowed to access more than one account.
- Monitor root user activity for possible compromise. The system needs to increase its alertness level, when root activities cross certain thresholds, such as specified number of active root processes.
- Monitor suspicious activities, e.g., a user executing unauthorized processes, user is present at a local and remote host concurrently as indicated by user's activity.
- Monitor for runaway processes. Certain non-daemon processes may be classified as runaway when the process keeps running even after the owner of the process has logged out.
- Tracking and tracing a suspicious user's activities throughout the monitoring domain.

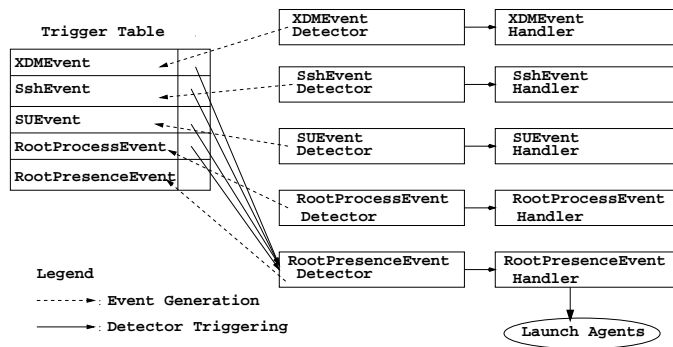


Figure 3. Trigger dependencies

We have conducted a number of monitoring experiments including the above mentioned scenarios. These detectors are built in steps as the events

generated by one detector can be used in a new compound detector. In this paper, we present the details of a detector for “suspicious root presence”, which illustrates the features of our system.

We consider root presence as suspicious when a new root process is detected without a corresponding legitimate login. If illegal processes, e.g. programs which should not be executing under root privileges, are found to be running as root, it is also classified as suspicious, and may indicate a root account compromise. Our *Process* and *Syslog* monitors look for processes running with root privileges and root logins respectively. The trigger dependencies are shown in Figure 3. The *XDMEvent* (X-Windows login), *SshEvent* and the *SUEvent* (“su” is executed) detectors are some of the possible ways a user could login as root. The *RootProcessEvent* detector checks for all new root processes. Any of these events, if generated, trigger the *RootPresence-Monitoring* detector. Here correlation takes place, and an event is generated if suspicious activity is found. This event is processed by its corresponding *handler*, which launches the *File Consistency* and *Fingerprinting* agents to that host. We also trace that user’s activities, and try to determine all information about that user, such as the machine he originally logged in from, and all processes run by that user.

#### 4 Related Work

Most of today’s monitoring systems rely on SNMP to collect data from various components<sup>15,16</sup>. Compared to existing SNMP and CMIP models, which support low level device management, our monitoring system can support abstractions for network-wide monitoring policies. In our approach, SNMP or CMIP can be integrated as one of the building-blocks for low level device monitoring. Another common approach used in today’s monitoring systems is to periodically execute scripts to monitor the status of a component or to process event data. Script based detection procedures tend to be cumbersome to install, debug, and to modify remotely.

#### 5 Conclusions

Our work can be compared with other distributed IDSs, such as Emerald<sup>17</sup>, GrIDS<sup>18</sup>, and NetSTAT<sup>19</sup>. Our focus, however, is also on the security of the monitoring infrastructure. We have presented a mobile agent based architecture for decentralized monitoring, aggregation and correlation of monitored data. The system provides utilities for installing new monitoring policies and supports installation of various response mechanisms by the system adminis-

trator. New monitoring functions can be launched automatically in response to certain events. The cooperating agents integrate events from different sources such as processes, syslogs, network traffic, file-system integrity checkers etc. New monitoring tools can be integrated into the system using agents as wrappers. In our approach, SNMP or CMIP can be integrated as one of the building-blocks for low level device monitoring. Currently we are integrating Snort<sup>20</sup>, a network based IDS, with our monitoring system. Our preliminary performance measurements indicate that on average the monitoring system uses between 1 to 5% of CPU on a Sun Ultra 10 with 256MB RAM running Solaris 5.8. The physical memory usage reaches upto 17MB, whereas the bare-bone JVM uses about 8MB with default JDK 1.3 settings.

## References

1. J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnegie Mellon University, January 2000. Available at <http://www.sei.cmu.edu/>.
2. Wayne Jansen, Peter Mell, Tom Karygiannis, and Don Marks. Applying Mobile Agents to Intrusion Detection and Response. National Institute of Standards and Technology Interim Report - 6416, October 1999. Available at URL <http://csrc.nist.gov/staff/mell/pmhome.html>.
3. William Wulf. Security experts: U.S. is unprepared for IT warfare. *IDG.net*, October 2001. Available at [http://www.idg.net/spc\\_711050\\_190\\_9-10025.html](http://www.idg.net/spc_711050_190_9-10025.html).
4. Gregory B. White, Eric Fisch, and Udo Pooch. Cooperating Security Managers: A Peer-Based Intrusion Detection System. *IEEE Network*, 10(1):20–23, January/February 1996.
5. William Lau, Sanjay Jha, and Mahbub Hassan. Current directions in active programmable network. In *Proceedings of the 9<sup>th</sup> IEEE International Conference on Networks*, pages 240–245, October 2001.
6. Jai Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni. An Architecture for Intrusion Detection using Autonomous Agents. Technical Report Coast TR 98-05, Department of Computer Sciences, Purdue University, 1998.
7. Mario Baldi, Silvano Gai, and Gian Pietro Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. In *Proceedings of the Workshop on Mobile Agents (MA'97) - LNCS 1219*, pages 13–26, April 1997.
8. P. Bellavista, A. Corradi, and C. Stefanelli. An Open Secure Mobile

- Agent Framework for Systems Management. *Journal of Network and Systems Management (JNSM)*, 7(3):323–339, September 1999.
9. Guy Helmer, Johnny Wongan Vasant Honavar, and Les Miller. Lightweight Agents For Intrusion Detection. Technical report, Department of Computer Science, Iowa State University, November 2000. <http://latte.cs.iastate.edu/Research/Intrusion/Papers.html>.
  10. Robert Pinheiro, Alex Poylisher, and Hamish Caldwell. Mobile Agents for Aggregation of Network Management Data. In *1<sup>st</sup> International Symposium on Agent Systems and Applications, and 3<sup>rd</sup> International Symposium on Mobile Agents*, pages 130–140, October 1999.
  11. A. Tripathi, T. Ahmed, S. Pathak, M. Carney, and P. Dokas. Paradigms for Mobile Agent-Based Active Monitoring. In *Network Operations and Management Symposium*, pages 65–78, April 2002.
  12. A. Tripathi, N. Karnik, M. Vora, T. Ahmed, and R. Singh. Mobile Agent Programming in Ajanta. In *Proceedings of the 19<sup>th</sup> International Conference on Distributed Computing Systems*, pages 190–197, May 1999.
  13. N. Karnik and A. Tripathi. Security in the Ajanta Mobile Agent System. *Software Practice and Experience*, 31(4):301–329, April 2001.
  14. G. H. Kim and E. H. Spafford. Experiences with Tripwire: Using integrity checkers for intrusion detection. In *Proceedings of the 3<sup>rd</sup> USENIX conference on Systems Administration, Networking and Security*, 1994.
  15. Raouf Boutaba, Karim El Guemhioui, and Petre Dini. An Outlook on Intranet Management. *IEEE Communications Magazine*, 35(10):92–99, October 1997.
  16. William Stallings. SNMP and SNMPv2: the infrastructure for network management. *IEEE Communications Magazine*, 36(3):37–43, March 1998.
  17. Phillip A. Porras and Peter G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20<sup>th</sup> National Information Systems Security Conference*, pages 353–365, October 1997.
  18. S. Staniford-Chen, S. Cheung, R Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS: A graph based intrusion detection system for large networks. In *Proceedings of the 19<sup>th</sup> National Information Systems Security Conference*, pages 361–370. National Institute of Standards and Technology, October 1996.
  19. G. Vigna and R.A. Kemmerer. NetSTAT: A Network-based Intrusion Detection System. *Journal of Computer Security*, 7(1):37–71, 1999.
  20. Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *13<sup>th</sup> Systems Administration Conference - LISA*, November 1999.