

Mobile Security Agents for Network Traffic Analysis*

Dipankar Dasgupta and Hal Brian
Intelligent Security Systems Research Group
Division of Computer Science
The University of Memphis
Memphis, TN 38152
Email: ddasgupt@memphis.edu

Abstract

This paper describes the implementation of a distributed agent architecture for intrusion detection and response in networked computers. Unlike conventional intrusion detection systems (IDS), this security system attempts to emulate mechanisms of the natural immune system using Java-based mobile software agents. These security agents monitor multiple levels (packet, process, system, and user) of networked computers to determine correlation among the observed anomalous patterns, reporting such abnormal behavior to the network administrator and/or possibly taking some action to counter a suspected security violation. The paper focuses on the design aspects of such an intrusion detection system by integrating different artificial intelligence techniques and a mobile agent architecture. Specifically, IBM's AgletsTM Software Development Kit (ASDK) is used as the base agent architecture, along with Adaptive Resonance Theory (ART-2) neural networks for network pattern classification, and a fuzzy logic controller for decision/action resolution. The feasibility and implementation of the mobile security agent system is demonstrated and some preliminary results are reported.

Key words: Intrusion Detection, Multi-Agent System, ART, Decision Support System, Fuzzy Controller, and Mobile agents.

1. Introduction

Anomaly detection is performed by detecting changes in the patterns of utilization or behavior of the system. This type of intrusion detection is performed by building a statistical model that contains metrics derived from system operation and flagging as intrusive any observed metrics that have a significant statistical deviation from the model [2,5]. In other words, an intrusion detection system based on anomaly detection uses a model of "normal" network behavior to compare to currently observed network behavior. Any behavior that varies

greatly from this model is considered a network intrusion and behavior closely matching the model is "normal".

In general, the normal behavior of a computing system can be characterized by observing its properties over time [12]. The problem of detecting anomalies (or intrusions) can be viewed as filtering non-permitted deviations of the characteristic properties in the monitored network system. This assumption is based on the fact that intruders' activities in some way must be different from the normal users' activities [11]. That assumption can lead to false-positives when any new behavior is considered anomalous and causes detection failure when intrusive behavior closely matches normal behavior.

Accordingly, one type of anomaly detection in use today is called Profile-Based Anomaly Detection (Figure 1), which focuses on characterizing the past behavior of individual users or related groups of users and then detecting significant deviations. A profile may consist of a set of parameters, so that deviation on just a single parameter may not be sufficient in itself to signal an alert.

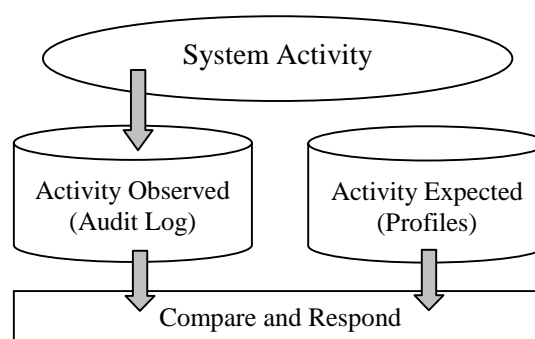


Figure 1. Profile-based anomaly detection [1]

The foundation of this approach is an analysis of audit records. The audit records provide input to the intrusion detection function in two ways. First, the designer must decide on a number of quantitative metrics that can be used to measure user behavior. An analysis of audit

* The paper is accepted for publication by the IEEE Computer Society Press in the proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX-II) to be held 12-14 June 2001 in Anaheim, California.

records over a period of time can be used to determine the activity profile of the average user. Thus, the audit records serve to define typical behavior. Second, current audit records are the input used to detect intrusion. That is, the intrusion detection model analyzes incoming audit records to determine deviation from average behavior. The main advantage of the use of statistical profiles is that a prior knowledge of security flaws is not required. The detector program learns what is “normal behavior” and then looks for deviations. The approach is not based on system-dependent characteristics and vulnerabilities. Thus, it should be readily portable among a variety of systems [20].

Another method, called Parameter Pattern Matching (Figure 2), involves the use of day-to-day operational experience as the basis for detecting anomalies. One of the more attractive characteristics of this method is that the administrators are not specifically targeting security issues. This introduces a more robust environment in which anomalies and patterns might be detected and matched. Such pattern matching constitutes an especially powerful processing approach because it provides an intrusion detection capability for attacks that might not be predictable [1].

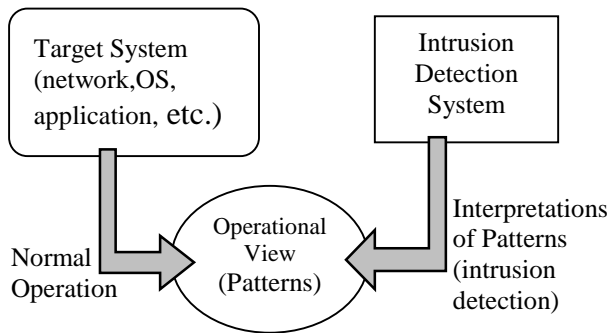


Figure 2. Parameter pattern matching [1]

This paper describes a mobile security agent architecture detecting coordinated and sophisticated attacks. The approach combines two anomaly detection methods by both profiling user behavior and also correlating it to network statistical behavior. The idea is that an intruder can be differentiated from a normal user by his activity and its associated impact on the system resources. This agent-based intrusion detection system (IDS) [11] attempts to emulate mechanisms of the natural immune system by detecting anomalies in a distributed manner.

2. Mobile Security Agent Architecture

The immunity-based IDS, named *SANTA* (Security Agents for Network Traffic Analysis) currently

implemented on *IBM's Aglets™* toolkit, which is composed of a set of Java-based mobile software agents that carry out specific tasks on the network and collaborate on the problem of network intrusion detection. A software agent can be defined as a software entity which functions continuously and autonomously in a particular environment and which is able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment.

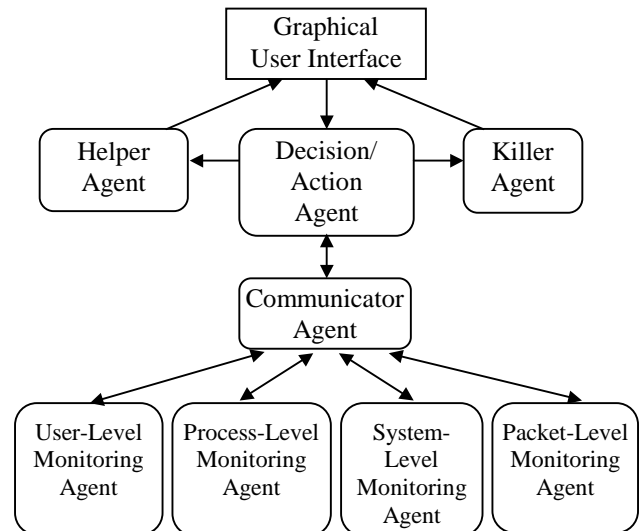


Figure 3. Mobile agent architecture

Ideally, an agent that functions continuously would be able to learn from its experience, to communicate and cooperate with other agents, and to move from place to place in doing so [5,7]. The proposed immunity-based agents roam around the machines (nodes or routers) and monitor the situation in the network (i.e., look for changes such as malfunctions, faults, abnormalities, misuse, deviations, intrusions, etc., as shown in Figure 4). These agents can mutually recognize each other's activities and can take appropriate actions according to the underlying security policies. Such an agent can learn and adapt to its environment dynamically and can detect both known and unknown intrusions [11].

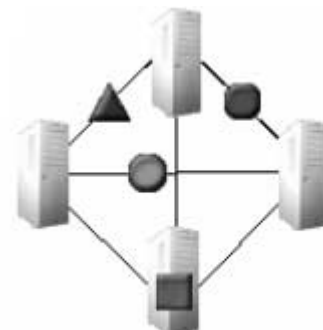


Figure 4. Different roaming security agents

This immunity-based IDS involves the collaboration of multiple autonomous software agents that detect deviations in the normal behavior of networked computers. Some agents monitor several parameters at multiple levels (packet, process, system, and user levels) to determine the correlation among the observed parameters during intrusive activities. Other agents wait for security violations to occur and then make a decision as to whether or not an action should be taken against the offending user or process. Accordingly, there are also some single-purpose agents that perform tasks such as communicating messages, reporting the status of the network, and/or killing processes. The roles and functionalities of the different types of agents are described below.

2.1. Monitoring Agents

These agents monitor networked computers by executing UNIX tools such as snoop, ps, and mpstat, looking for deviations in the learned normal behavior. For each network host, there is an associated monitoring agent for each of the four levels: packet level, process level, system level, and user level. At the packet level, an agent detects changes in the numbers and sizes of packets for different protocols. At the process level, a different agent detects unusual process memory allocation, priority, CPU usage, etc. At the system level, an agent looks at overall system memory, CPU, and I/O usage. At the user level, an agent scans the UNIX syslog file for login failures, attempts to gain root access, etc. All of these agents report anomalous behavior to a Decision/Action Agent (D/A Agent) for further processing. Figure 5 illustrates the process that Monitoring Agents follow to learn the behavior of monitored parameters using ART Neural Networks (NN).

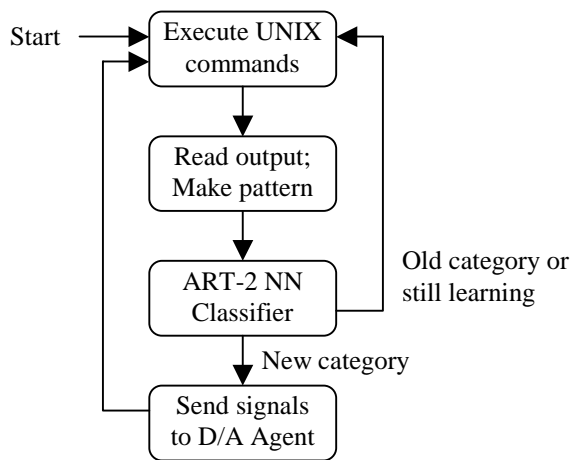


Figure 5. Monitoring agent process

2.2. Communicator Agents

These agents pass messages between agents. The Aglets Software includes these Messenger Agents as a primary feature.

2.3. Decision/Action Agents

These agents make decisions as to whether an action should be taken on behalf of the system administrator based on the information from the Monitoring Agents. Each of them has a fuzzy logic controller component in order to determine the severity of the anomaly and the age of such previous incidents to determine whether the D/A Agent further activates one or more Response Agents--Helper Agents and Killer Agents (Figure 6).

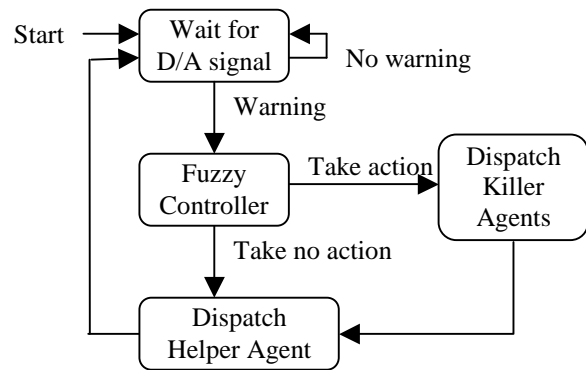


Figure 6. Decision/action agent process

2.4. Helper Agents

These agents provide status information to the system administrator's Graphical User Interface (GUI). They are activated by the Decision/Action Agent when a warning is received from the Monitoring Agents or when a Killer Agent has been dispatched.

2.5. Killer Agents

These agents terminate processes (using the UNIX kill -9 process id command) that are responsible for intrusive behavior on the network. The Decision/Action Agent dispatches a Killer Agent when the Threat Level determined by its Fuzzy Logic Controller is Medium-High or High. Once the process is terminated, the Killer Agent reports the action to the GUI using a Helper Agent (Figure 6).

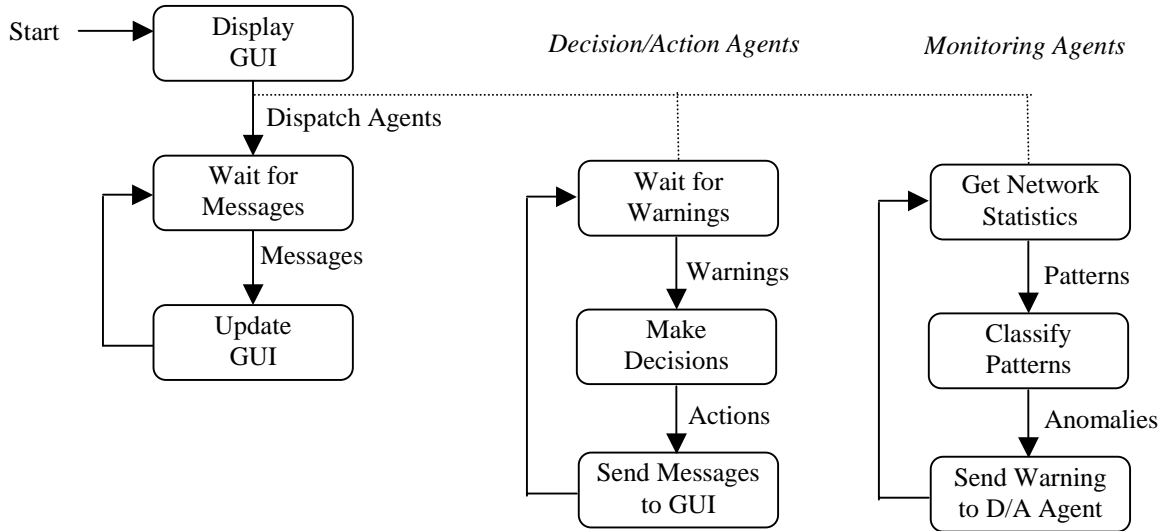


Figure 7. Overall line diagram showing different processes of SANTA

The overall sequence of steps of SANTA is shown in Figure 7. Once the agents are dispatched to the desired host on the network, the Graphical User Interface (Figure 13) waits for messages from the agents to update its display. The Decision/Action Agents wait for warning signals from the Monitoring Agents and make decisions based on the information regarding violations. Any actions are relayed to the GUI for display. The Monitoring Agents begin immediately sensing network status and classifying them into distinct categories using the ART neural network. After the training is completed, these agents report patterns that do not fit into known categories as anomalies to the Decision/Action Agent.

3. Implementation Details

The IBM Aglets™ Software Development Kit (ASDK version 1.0.3) was selected for the agent-programming environment due to its ease of use, and associated rapid development time of agents. The software was downloaded from IBM’s Tokyo Research Laboratory website (www.trl.ibm.com/aglets). Java 1.1 was required for this implementation for compatibility reasons. After installing Java and Aglets on a UNIX server, the Tahiti Aglet User Interface was launched to host the mobile agents. Two instances of Tahiti were used to simulate more than one server on the network with different port numbers distinguishing the two Aglet Server addresses (Figure 8).

The first Tahiti interface in Figure 8 shows that the server called test, port 434, is hosting the main SANTA Aglet named Santa.Santa. It contains the GUI and related message sending and receiving methods. Initially, this

Aglet is launched using the “Create” button and selecting the appropriate Java class. The main administrator GUI

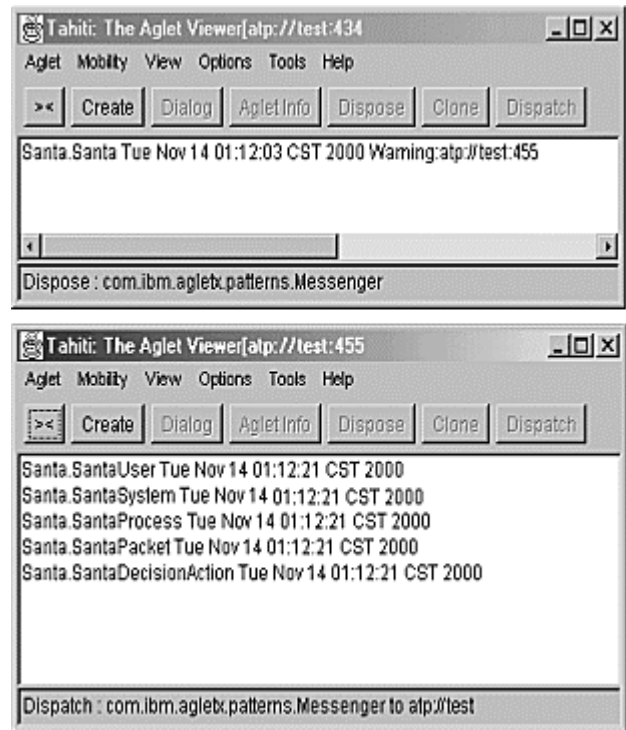


Figure 8. Tahiti Aglet User Interface

immediately appears (as shown in Figure 13) and the user selects a host computer for the Decision/Action and Monitoring Agents. The Aglets are then dispatched by the user to initiate the intrusion detection process. The

second Tahiti interface in Figure 8 displays the Decision/Action Agent and all four Monitoring Agents residing on the Aglet Server called test, port 455.

Security properties must also be specified in Tahiti to allow read access for any audit logs or data files and execute access for any UNIX commands [4]. Mobile agents dispatched from other Tahiti interfaces are considered untrusted for security purposes.

3.1 Decision support components

Some Monitoring Agents store network, system, or user behavior patterns to serve as a knowledge base or model of known “normal” behavior. When learning of “normal” behavior has ceased, the agent compares current network, system, or user behavior with its knowledge base of “normal” patterns. Any patterns that do not closely match previously seen patterns are considered anomalous and are reported to other agents for possible action against the user or process.

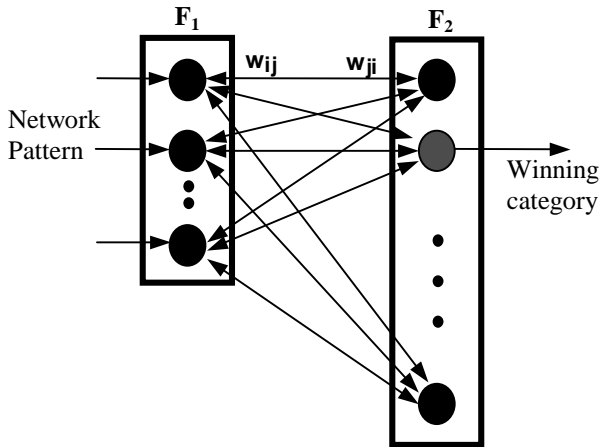


Figure 9. ART-2 Neural Network

3.1.1 ART-2 Neural Network Classifier. The Adaptive Resonance Theory (ART) neural network classifier (developed by Grossberg [8,15]) was chosen due to its ability to group presented patterns into categories without human supervision. ART is one type of an unsupervised neural network that uses competitive learning (Figure 9). A pattern that does not closely match any of the known categories either causes the network to add a new category during the learning phase (Figure 10) or identifies the pattern as anomalous during the testing phase (Figure 11).

ART-2 networks self-organize stable recognition categories in response to arbitrary sequences of analog (gray-scale, continuous-valued) input patterns. ART networks encode new input patterns, in part, by changing the weights, or long-term memory (LTM) traces, of a

bottom-up adaptive filter. This filter is contained in pathways leading from a feature representation field (F_1) to a category representation field (F_2) whose nodes undergo cooperative and competitive interactions. In an ART network there is a second, top-down adaptive filter that leads to the crucial property of code self-stabilization. Such top-down adaptive signals play the role of learned expectations in an ART system. They enable the network to carry out attentional priming, pattern matching, and self-adjusting parallel search [15].

In order to cope with arbitrary sequences of analog input patterns, ART-2 architectures embody solutions to a number of design principles, such as stability-plasticity tradeoff, the search-direct access tradeoff, and the match-reset tradeoff. A parallel search scheme updates itself adaptively as the learning process unfolds, and realizes a form of real-time hypothesis discovery, testing, learning, and recognition. After learning self-stabilizes, the search process is automatically disengaged. Thereafter input patterns directly access their recognition codes without any search. Thus, recognition time for familiar inputs does not increase with the complexity of the learned code. A novel input pattern can directly access a category if it shares invariant properties with the set of familiar exemplars of that category. The architecture’s global design enables it to learn effectively despite the high degree of nonlinearity of such mechanisms [8].

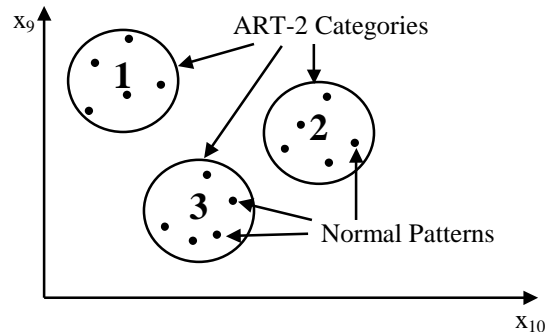


Figure 10. ART-2 categories created in ten-dimensional hyperspace

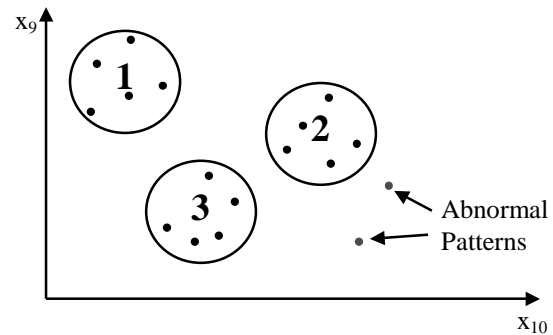


Figure 11. Abnormal patterns falling outside known “normal” behavior categories

3.1.2. Fuzzy Controller. A fuzzy controller [21] was developed for the Decision/Action Agents that must make decisions and possibly take action based on anomalous behavior. These agents receive reports of security incidents, including the severity of the event, from the Monitoring Agents and the cumulative totals are adjusted for age. Together, these incident characteristics are fed to a fuzzy controller, and a decision is made to take some action, such as terminate a process, or do nothing, based on the controller output.

There were five fuzzy sets created representing Low, Low-Medium, Medium, Medium-High, and High threat levels (Figure 12(a)). 5^4 or 625 fuzzy rules were defined to govern the controller's decisions. For example, one such rule was "if System Threat is Low and User Threat is Low and Process Threat is Low and Packet Threat is Low, then Threat Level is Low." Based on the inputs from these four monitored levels of the network, the degree of membership to each set was calculated, and the union of the five resulting fuzzy sets was determined (Figure 12(b)) using the 2^4 or 16 active fuzzy rules. A defuzzification method was applied to the set union to find the center of gravity of the set, yielding the actual Threat Value. If this value exceeded the Threat Threshold (>0.5 , Medium-High or High), the controller dispatched an agent to kill the associated process, if one existed, or warned the network administrator of a high threat level. Otherwise, the controller took no action. Figures 12(a) and 12(b) show the membership functions used to implement the Fuzzy Controller.

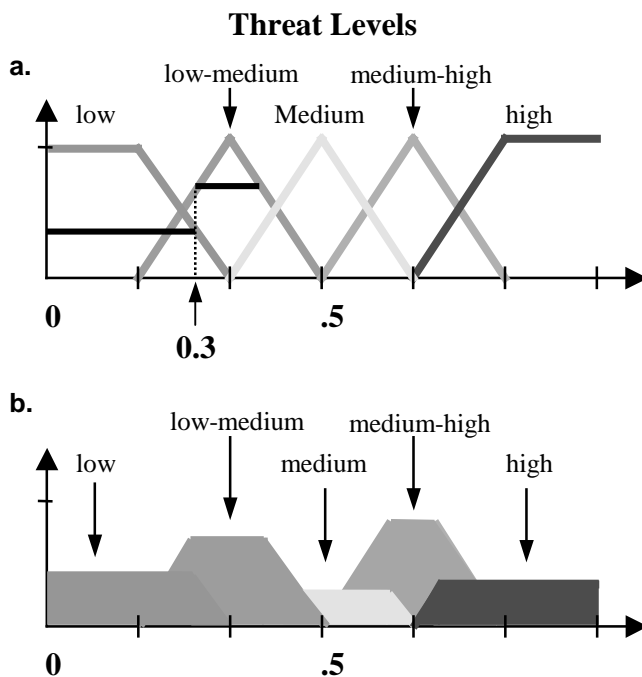


Figure 12. a. Five Fuzzy Sets, b. Union of Sets

4. Experimentation

Experiments were conducted in the network security lab to test SANTA's ability to learn normal network behavior and detect anomalous behavior. The initial results were very promising with successful classification of network patterns during the learning phase and detection of known anomalies and intentional violations during the testing phase (shown in Figure 13). Factors determining SANTA's sensitivity to anomalies were the length of training time (learning phase) of Monitoring Agents, ART-2 thresholds, and the age function of the fuzzy controller. These parameters were repeatedly adjusted until a stable combination was found.

Table 1 shows the rate at which the Monitoring Agents created categories of network patterns under average server load assuming that all network behavior was "normal". Ten trials were performed for each agent using a fixed pattern ratio. For example, the System-Level Monitoring Agent read 500 patterns resulting in seven ART-2 categories. The agent then classified another 500 patterns into six old categories during the testing phase. Five patterns, representing the false-positives, did not fit into any of the original seven categories.

Training time was restricted to short intervals during system development and then later increased incrementally during system testing. The longer the agents trained, the more patterns were stored in the ART-2 network, and the greater the sensitivity to true anomalies while false-positives were decreased (Table 2).

ART-2 thresholds determined the size of the ART categories and ultimately, the sensitivity to small differences in network patterns. Higher thresholds meant more refined categories and more precise detection of anomalies. The disadvantage, though, was an increase in the number of categories and false-positives. Since the agent's memory was limited, too high a threshold caused the agent to run out of available categories before all normal behavior was learned. On the other hand, too small a threshold lead to broad categories and detection failure (false-negatives), while using very few of the available categories. The threshold used for all three agents during actual implementation was 0.99, forcing highly precise categorization of network behavior patterns and limiting false-negatives.

The age function in the fuzzy controller determined the rate at which the threat levels of previous incidents were degraded over time. Faster degradation limited the correlation of current incidents to others occurring at about the same time while slow degradation allowed correlation of events occurring over a wider period of time. The function used in SANTA degraded the threat levels at 0.01 per second, ensuring total degradation after 100 seconds.

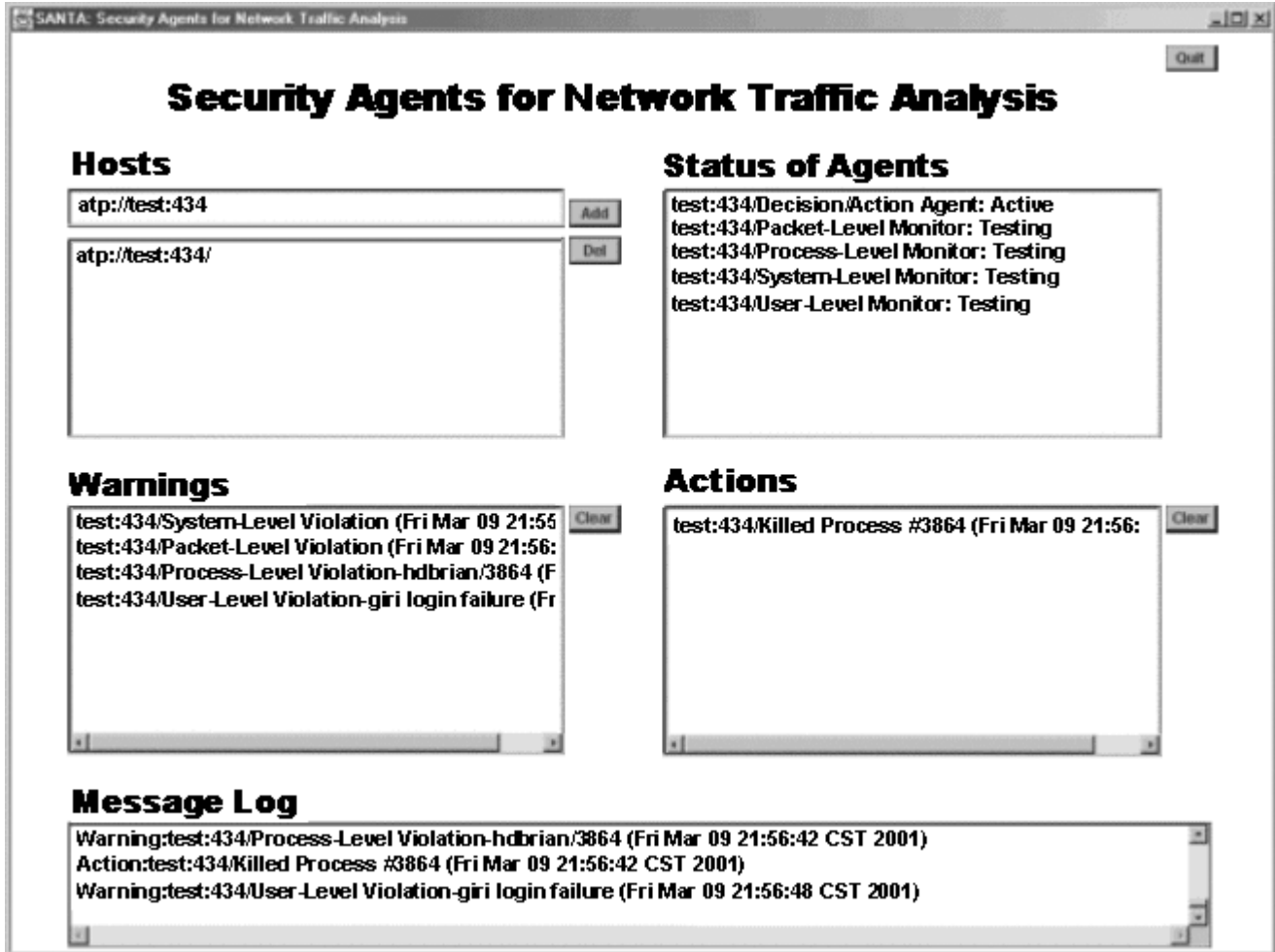


Figure 13. SANTA's Graphical User Interface

Table 1. Learning "Normal" Network Behavior using ART-2 Classifiers

Trial #	System-Level Monitoring Agent (500/500 Pattern Ratio)			Process-Level Monitoring Agent (15000/15000 Pattern Ratio)			Packet-Level Monitoring Agent (50/50 Pattern Ratio)		
	Learning Phase Categories	Testing Phase Categories	False- Positives	Learning Phase Categories	Testing Phase Categories	False- Positives	Learning Phase Categories	Testing Phase Categories	False- Positives
1	11	9	5	8	7	0	9	2	1
2	7	7	35	9	8	0	4	4	2
3	7	6	3	9	8	0	5	5	1
4	6	6	1	8	6	1	5	5	0
5	6	4	0	8	7	2	5	3	2
6	6	5	0	9	8	0	2	2	2
7	8	5	0	9	8	0	6	3	0
8	8	4	2	8	7	0	6	4	0
9	5	4	2	9	8	0	6	3	0
10	7	5	1	9	8	0	6	4	4
Average	7.1	5.5	4.9	8.6	7.5	.3	5.4	3.5	1.2

Table 2. Effect of Pattern Ratio on Learning Phase Categories and False-Positives

System-Level Monitoring Agent				Process-Level Monitoring Agent				Packet-Level Monitoring Agent			
Pattern Ratio	Learning Phase Categories	Testing Phase Categories	False-Positives	Pattern Ratio	Learning Phase Categories	Testing Phase Categories	False-Positives	Pattern Ratio	Learning Phase Categories	Testing Phase Categories	False-Positives
250/500	7	5	42	1000/15000	8	7	1	25/50	3	3	6
500/500	7	6	5	5000/15000	9	8	0	50/50	5	4	1
750/500	8	4	1	15000/15000	9	8	0	75/50	9	7	0
1000/500	10	5	0	25000/15000	9	8	0	100/50	11	8	0

5. Conclusion

When a hacker attacks a system, the ideal response would be to stop his activity before he can cause any damage or gain access to any sensitive information. This would require recognition of the attack as it takes place. Different models of intrusion detection have been developed [14,18], and much IDS software is available for use. Commercial IDS products such as NetRanger, RealSecure, and Omniguard Intruder Alert work on attack signatures. These signatures needed to be updated by the vendors on a regular basis in order to protect from new types of attacks. However, no detection system can catch all types of intrusions and each model has its strengths and weaknesses in detecting different violations in networked computer systems. Recently, researchers started investigating techniques like artificial intelligence [9,13,16], autonomous agents [5,6,10] and mobile agent architectures [3,17] for detecting intrusion in network environment.

Most existing intrusion detection systems either use packet-level information or user activities to make decisions on intrusive activities [12,19]. In this paper, an agent-based intrusion detection system (SANTA) is described that can simultaneously monitor network activities at different levels (such as packet level, process level, system level, and user level). This SANTA system represents a novel approach to distributed intrusion detection. The system emulates some mechanisms of the human immunity system and features distributed identification of anomalies and decentralized control of decisions and responses to those anomalies. The IBM Aglets agent software provides a suitable environment for hosting the mobile autonomous agents that comprise this intrusion detection system. Agents can move throughout the network observing network behavior patterns and communicating any anomalies to other agents for action. The ART-2 neural network classifier is an ideal learning mechanism for Monitoring Agents. Observed network patterns can be classified into categories during a learning phase without loss or degradation to previously created categories. Patterns that fail to fit into known categories

during the testing phase are assumed to be anomalous. A fuzzy controller takes all anomaly reports as input and determines the current Threat Level. If the threat is Medium-High or High, the Decision/Action Agent will take action to terminate the associated process. Though the long-term plan of this project is to develop immunity-based mobile agent architecture from design principles, the use of *IBM's AgletsTM* in our current implementation is a proof-of-concept for immunity-based intrusion detection system framework [11].

6. References

- [1] Amoroso, E., *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*, Intrusion.Net Books, Sparta, New Jersey, 1999.
- [2] Allen, J. et al., "State of the Practice of Intrusion Detection Technologies", *Technical Report* (No. CMU/SEI-99-TR-028), January 2000.
- [3] Asaka, M., S. Okazawa, A. Taguchi, and S. Goto, "A Method of Tracing Intruders by Use of Mobile Agents", *INET'99*, June 1999.
- [4] Axelsson, S., U. Lindqvist, U. Gustafson, and E. Jonsson, "An Approach to UNIX Security Logging", *Technical Report*, IEEE Network, 1996.
- [5] Balasubramanian, J., J. Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. "An Architecture for Intrusion Detection Using Autonomous Agents", *COAST Technical Report 98/5*, Purdue University, 1998.
- [6] Barrus, J. and N. Rowe, "A Distributed Autonomous-Agent Network-Intrusion Detection and Response System", *Proceedings of the Command and Control Research and Technology Symposium*, Monterey, CA, June 1998.
- [7] Berenji, H. and D. Vengerov, "Learning, Cooperation, and Coordination in Multi-Agent Systems", *Technical Report*, IIS-00-10, 2000.
- [8] Carpenter, G. and S. Grossberg, "ART 2: Self-organization of Stable Category Recognition Codes for Analog Input Patterns", *Applied Optics*, Vol. 26, No. 23, December 1987.

[9] Carver, C., J. Hill, J. Surdu, and U. Pooch, "A Methodology for Using Intelligent Agents to Provide Automated Intrusion Response", *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, West Point, NY, June 2000.

[10] Crosbie, M. and E. Spafford, "Defending a Computer System using Autonomous Agents", *Proceedings of the 18th National Information Systems Security Conference*, October 1995.

[11] Dasgupta, D., "Immunity-Based Intrusion Detection Systems: A General Framework", *Proceedings of the 22nd National Information Systems Security Conference (NISSC)*, <http://issrl.cs.memphis.edu/nissc-99.pdf>, October 18-21, 1999.

[12] Debar, H., M. Dacier, and A. Wepszi, "A Revised Taxonomy for Intrusion Detection Systems", *Technical Report*, Computer Science/Mathematics, 1999.

[13] Frank, J., "Artificial Intelligence and Intrusion Detection: Current and Future Directions", *Proceedings of the 17th National Computer Security Conference*, October 1994.

[14] Frincke, D., "Balancing Cooperation and Risk in Intrusion Detection", *ACM Transaction on Information and System Security*, Vol. 3, No. 1, February 2000.

[15] Grossberg, S., ed., *Neural Networks and Natural Intelligence*, MIT Press, Cambridge, Massachusetts, 1988.

[16] Helmer, G., J. Wong, V. Honavar, and L. Miller, "Intelligent Agents for Intrusion Detection", *Proceedings, IEEE Information Technology Conference*, pages 121-124, Syracuse, NY, September 1998.

[17] Jansen, W., P. Mell, T. Karygiannis, and D. Marks, "Mobile Agents in Intrusion Detection and Response", *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada, June 2000.

[18] Lane, T. and C.E. Brodley, "Temporal Sequence Learning and Data Reduction for Anomaly Detection", *ACM Transaction on Information and System Security*, Vol. 2, No. 3, August 1999.

[19] Mukherjee, B., L.T. Heberline, and K. Levit, *Network Intrusion Detection*, IEEE Network, 1994.

[20] Stallings, W., *Network Security Essentials: Applications and Standards*, Prentice Hall, Upper Saddle River, New Jersey, 1999.

[21] Zadeh, L., "Fuzzy Logic, Neural Networks and Soft Computing", *Communications of the ACM*, Vol. 37, No. 3, pp. 77-84, 1994.

Acknowledgements

The authors would like to thank the University of Memphis Intelligent Security Systems Research Laboratory members for their useful comments and suggestions, and especially Dan Lebowitz for providing vital technical assistance. This work is partly funded by the Defense Advanced Research Projects Agency (DARPA) under the contract no. F30602-00-2-0514.