Software Agents and Computer Network Security

Pikoulas J, Mannion M, Buchanan W
Napier University,
2 19 Colinton Road,
EH14 1DJ, Edinburgh
Scotland, UK
Email: {johnp,bill}@dcs.napier.ac.uk
Email: m.mannion@napier.ac.uk

ABSTRACT

Preventing unauthorised access to corporate information systems is essential for many organisations. To address this problem we built a security enhancement software system using software agents, in which a core software agent resides on a server and user end software agents reside at each user workstation. By downloading a pattern of typical user behaviour and rules governing invalid behaviour from a core agent to each user end agent, all decisions and actions about atypical or invalid user behaviour can be taken by a user agent. This permits security detection to continue even when the core agent fails to operate

1 Introduction

Computer network security is concerned with preventing the intrusion of an unauthorised person into a computer network. As computer connectivity increases, computer network security becomes more complex. Intrusion [1] is any set of actions that attempt to compromise the integrity, confidentiality or availability of a computer system resource (for example, unauthorised distribution of sensitive material over the Internet).

Security enhancement software usually consists of programs written by computer network administrators in an operating system script language to ensure that normal usage patterns occur. The most commonly used security-enhancement method is a user name and a password checking facility. This is a simple and robust system but has drawbacks. One drawback is that if someone manages to overcome the user name and password barrier there are few means of preventing the system being modified or destroyed. A second drawback is that whilst some of these programs can be written in advance (proactively), it is difficult to anticipate every possible case of abnormal behaviour. In addition, most organisations cannot afford to have an administrator working full time on writing security programs. One solution for tackling these problems is to create and maintain historical user profiles of normal usage and compare them with current usage, and monitor the differences.

Most users have normal usage patterns. These patterns occur on a system wide level (for example, the type and mix of jobs being run) and on a user level (for example, average job length, type of job user run, normal user usage hours). An historical user profile is the synthesis of these patterns over a defined period.

A profile of a user can be created from an audit log file. An audit log file is a text file that a system maintains and updates daily. It records every action by every user that has logged on to the server.

During each login session, a user's profile is reviewed to ensure that their behaviour pattern lies within their permitted behaviour. If a user pattern does not match their profile, then appropriate actions are taken, the most extreme of which might be to automatically terminate the user's session. There are several issues to be resolved when applying the profiling technique including:

- Deciding the format of the profile.
- How to change the profile.
- How to compare a profile with current usage so that it is clear whether any differences highlight legal or illegal behaviour.
- How often to update a user's profile.
- The storage requirements of user's profile.

We tackle some of these issues by using a software agent system to monitor and predict the behaviour pattern of a user, and then comparing this pattern with the user's historical profile.

We applied this paradigm to security enhancement software for a Windows NT client server environment. A core agent resides on one server of the system. User agents reside in each user workstation, monitor the various actions of a user, and take appropriate action if a user exhibits invalid behaviour. The user agent then informs the core agent. We report

0-7695-0604-6/00 $10.00

the results of experiments in which software engineering students attempted to breech the security of the networked system.

This paper is organised as follows. Section 2 explains common computer network security techniques for constructing patterns of typical user behaviour and previous work in network security using software agents. Section 3 describes our software agent security enhancement system. Finally, our experimental results are discussed in Section 4.

2 Computer Network Security Systems

Computer network security programs can be categorised as follows [2]:

- Security enhancement software enhances or replaces an operating system's built-in security software (for example, Mangle It, Passwd+, Shadow).
- Authentication and encryption software encrypts and decrypts computer tiles (for example, Kerberos, MD5, RIPEM, and TIS Firewall Toolkit).
- Security monitoring software monitors different operations of a computer network and outputs the results to system administrators (for example, Abacus Sentry, COPS, Tripwire, Tiger).
- Network monitoring software monitors user's behaviour or monitors incoming or outgoing traffic (for example, Argus, Arpwatch, ISS).
- Firewall software resides in the Internet entrance of a computer network, and checks all incoming network traffic for valid Internet Protocol (IP) addresses, and also TCP connections.

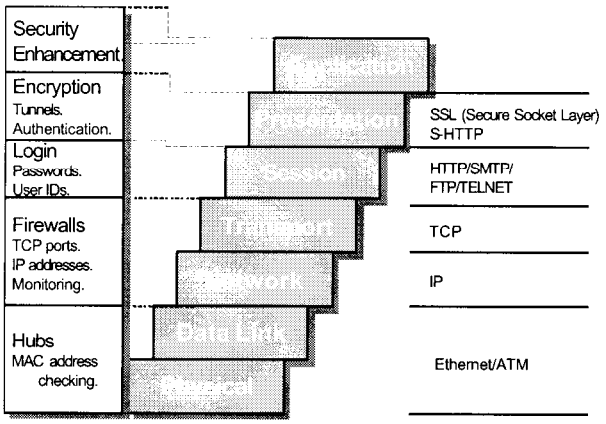

Figure 1 OSI layer classification

Figure 1 shows the contribution each of these programs makes with respect to the Open Systems Interconnection Reference Model (OSI Model). Typical network attacks [3] are:

- **IP spoofing attacks.** This is where the hacker steals an authorised IP address. Typically, it is done by determining the IP address of a computer and waiting until there is no-one using that computer, then using the unused IP address. Several users have been accused of accessing unauthorised material because other users have used their IP address. A login system which monitors IP addresses and the files that they are accessing over the Internet cannot be used as evidence against the user, as it is easy to steal IP addresses.
- **Packing sniffing.** This is where the hacker listens to TCP/IP packets which come out of the network and steals the information in them. Typical information includes user logins, e-mail messages, credit card number, and so on. This method is typically used to steal an IP address, before an IP spoofing attack. A hacker listens to a conversation between a server and a client. Most TELNET and FTP programs actually transmit the user name and password as text values; these can be easily viewed by a hacker.
- **Passwords attacks.** This is a common weak-point in any system, and hackers will generally either find a user with a easy password (especially users which have the same password as their login name) or will use a special program which cycles through a range of passwords. This type of attack is normally easy to detect. The worst nightmare of this type of attack is when a hacker determines the system administrator password (or a user who has system privileges). This allows the hacker to change system set-ups, delete files, and even change user passwords.
- **Sequence number prediction attacks.** Initially, in a TCP/IP connection, the two computers exchange a start-up packet which contains sequence numbers. These sequence numbers are based on the computer's system clock and then run in a predictable manner, which can be determined by the hacker.
- **Session hi-jacking attacks.** In this method, the hacker taps into a connection between two computers, typically between a client and a server. The hacker then simulates the connection by using its IP address.
- **Shared library attacks.** Many systems have an area of shared library files. These are called by applications when they are required (for input/output, networking, graphics, and so on). A hacker may replace standard libraries for ones that have been tampered with, which allows the hacker to access system tiles and to change file privileges. A hacker might tamper with dynamic libraries (which are called as a program runs), or with static libraries (which are used when compiling a program). This would allow the hacker to possibly do damage to the local computer, send a11 communications to a remote computer, or even view everything that is viewed on the user screen. The hacker could also introduce viruses and cause unpredictable damage to the computer (such as remotely rebooting it, or crashing it at given times).
- **Social engineering attacks.** This type of attack is aimed at users who have little understanding of their computer system. A typical attack is where the hacker sends an email message to a user, asking for their password. Many unknowing users are tricked by this attack. From the initial user login, the hacker can then access the system and further invade the system.
- **Technological vulnerability attack.** This normally in-

volves attacking some part of the system (typically the operating system) which allows a hacker to be access to the system. A typical one is for the user to gain access to a system and then run a program which reboots the system or slows it down by running a processor intensive program. This can be overcome in operating systems such as Microsoft Windows and UNIX by granting reboot rights only to the system administrator.

- **Trust-access attacks.** This allows a hacker to add their system to the list of systems which are allowed to log into the system without a user password. In UNIX this file is the ".rhosts" (trusted hosts) which is contained in the user's home directory. A major problem is when the trusted hosts file is contained in the root directory, as this allows a user to log in as the system administrator.

In choosing a combination of the computer network security programs that mentioned above, the dominant issues are cost, the desired level of security and the characteristics of the existing operating system environment. Three techniques for illegal behaviour detection are commonly used in computer network security programs [4].

- Statistical anomaly detection.
- Rule based detection.
- Hybrid detection, an amalgam of statistical anomaly detection and rule based detection.

These techniques can be applied to all five categories of computer security program.

2.1 Statistical anomaly detection

Statistical anomaly detection systems analyse audit-log data to detect abnormal behaviour. A profile of expected online behaviour for a normal user is predefined and derived from how an organisation expects a user to behave and from a system administrator's experience of the way a user is expected to use the resources of a system. Typically, periodically, analysing audit logs and looking for statistical patterns of events for typical operations establish usage patterns for an average user. These patterns are compared to the user's profile.

Debra Anderson led a project called Safeguard [5], to adapt the NIDES statistical anomaly-detection subsystem to profile the behaviour of individual applications. Statistical measures were customised to measure and differentiate the proper usage of an application from an inappropriate usage. Under the Safeguard model, a statistical score is assigned to the operation of applications and represents the degree to which current behaviour of the application corresponds to its established pattern of operation. The Safeguard effort demonstrates the ability of statistical profiling tools and clearly differentiates the scope of execution among general-purpose applications. It also showed that statistical analyses could be very effective in analysing activities other than individual users; by instead monitoring applications, the Safeguard analysis greatly reduced the required number of profiles and computational requirements, and decreased the typical false-positive and false-negative ratios. These results suggest the possible utility of performing statistical analyses on activities at higher layers of abstraction.

Supervisors of the system are warned of a possible intrusion when a profile is different to a usage pattern. The major drawback of this technique is that it cannot predict extreme changes in user behaviour. If a user radically changes their habits, the assumption is that the user is trying to harm the system.

2.2 Rule-based detection

In rule-based detection systems a set of rules of typical illegal user behaviour are created. The rules are formed by analysing previous different patterns of attack. A rule based detection system analyses audit-log data of a particular user and compares it with the rules. The drawback of this system is that the basic rules are predefined by system administrators, and can not detect new attack techniques. If a user exhibits behaviour that is not prescribed by the existing rules, the user can harm the system without being detected.

The IDES system [6] is security enhancement software that stores knowledge about a system's known vulnerabilities, its security policies and knowledge of past intrusions. The information it uses to determine network state is limited to packet header data. Since this system does not examine the whole packet, it can miss critical information about the nature of the data that goes throughout the network. It also scales very poorly where many machines are on a high-speed network.

Kumar and Spafford's [7] security enhancement software uses pattern matching. Attacks can be classified as patterns, which match against occurrences (status of the system at that moment) in the system. These patterns can encode dependencies between system conditions and temporal conditions.

Crosbie and Spafford's [8] system security enhancement software uses autonomous agents, The agents are trained to detect anomalous activity in network system traffic. A drawback to this approach is that the system requires considerable training by a human operator to be trained before it becomes effective.

2.3 Hybrid Detection

Hybrid detection systems are a combination of statistical anomaly detection and rule-based detection systems. Typically, rules are used to detect known methods of intrusion and statistical based methods are used to detect new methods of intrusion.

The CMDS (Computer Misuse Detection System) [9] is security-monitoring software that provides a way to watch for intrusions even in switched networks. CMDS detects and thwarts attempted logins, file modifications, Trojan horse installation, changes in administrative configurations and many other signs of intrusion. In addition, CMDS constantly monitors for the difficult detection problems like socially engineered passwords, trusted user file browsing and data theft that might indicate industrial espionage. CMDS supports a wide variety of operating systems and application programs. The drawback of this system is that it uses statistical analysis to make additional rules for the system. That can be a draw-

back, because it can only detect attack patterns that had being used in the past and being identified as attack patterns, or predefined by the system operators. It also generates long reports and graphs of the system performance that still requires the presence of an expert to read and analyse their meaning.

In most cases, the implementation of the three techniques above has been achieved by locating the security enhancement software on a centralised server. When this software crashes or is breeched, network security is at risk. To address this we have built a security enhancement environment in which security management is dispersed across the network using software agents.

3 Software agents and their use for security enhancement

Software agents have the following properties [IO]:

- **Autonomy.** They can operate without the direct intervention of humans.
- **Cooperativity.** They can cooperate with other agents.
- **Reactivity.** They can perceive their environment and respond in a timely fashion to changes that occur in it.
- **Proactivity.** They can detect patterns in their environment, and exhibit goal directed behaviour by taking the initiative.

Scvcral sccurity enhancement approaches deploy agent technology as a tool for detecting abnormal behaviour in the system AID (Adaptive Intrusion Detection system [11]) consists of a core agent running on a server and a user agent running on each client workstation. The core agent hosts a manager and an expert system. The manager continually polls each client requesting data about current user behaviour. This data is then examined by the expert system, which contains rules governing valid user behaviour. If invalid behaviour is detected the manager takes an appropriate course of action.

In the Autonomous Agents for Intrusion Detection [12] system, a core agent hosts a manager and an expert system but at each client workstation, there are several user agents and a monitor. The behaviour of each user is recorded by the user agents and sent by the monitor to the core agent. The core agent determines if invalid behaviour has been detected but sends instructions to the monitor to take an appropriate course of action.

In our approach all decisions and actions about invalid user behaviour are taken by the user agent at the client workstation In addition, we combine a set of rules governing invalid behaviour with a profile of typical user behaviour.

4 Software agent computer intrusion system

We built intelligent agent security enhancement software system, in which a core software agent resides on one server in a Windows NT network system and user end software agents reside in each user workstation. The software for each type of agent was written in SUN Java JDK Version 1.2 on a Microsoft Windows NT Version 4 environment running over a 10/100 Mbps network. There was 1 server and 10 clients.

Figure 2 shows a core agent communicating with many user agents and Figure 3 shows the steps taken by the user agent. A communication thread is a unique process that the core agent creates to transmit data to the user end agent in response to message transmitted from the user end agent. Unique processes enable the core agent to communicate with each user agent effectively and efficiently thereby enabling a fast response to network monitoring. Once the core agent has responded to a user agent, the process is killed.

The system uses a hybrid detection technique. Invalid behaviour is determined by comparing a user's current behaviour with their typical behaviour and by comparing their current behaviour with a set of general rules governing valid behaviour formed by systems administrators. Typical behaviour is contained in a user historical profile. The core agent builds a user's historical profile from a statistical analysis of an audit-log tile of the system server using Marcov chains [13]. In this work, the period covered by the audit-log tile was one month.

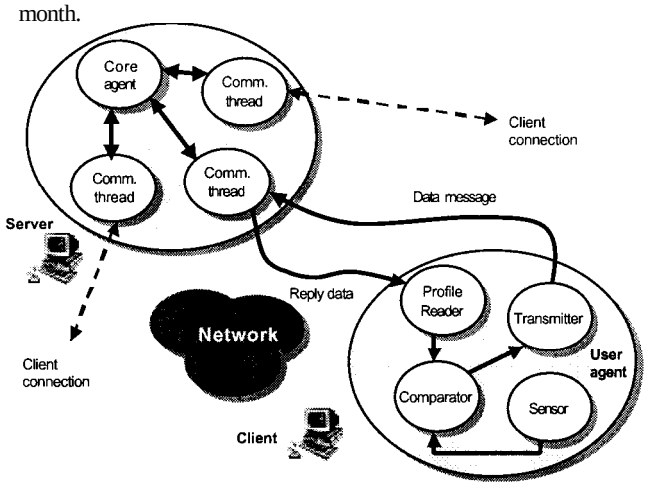

Figure 2 Software agent security enhancement system

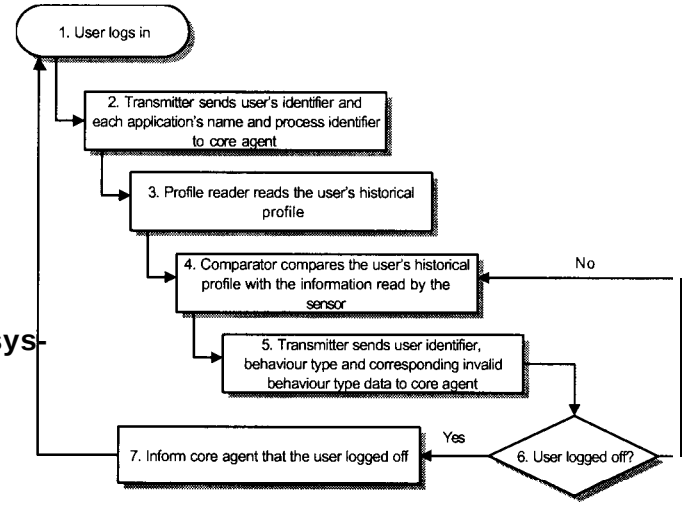

Figure 3 User agent flow diagram

Figure 3 User agent flow diagram

The user agent software has four components: a transmitter; a sensor; profile reader; and a comparator. The sensor monitors the various software applications (for example, a word processor, a spreadsheet) that are currently being run by the user on that workstation. When a user logs in the sensor polls the user's activity every five seconds and records the user's identifier and each application's name and process identifier. After the first polling by the sensor, the transmitter sends this information to the core agent. The core agent responds by sending a user historical profile. With an audit-log file for a period of one month, we observed that the size of an average user profile was between 400KB and 600KB. The download time for this size of file was about three to five seconds.

The profile-reader reads the user's historical profile. The comparator compares the user's historical profile with the information read by the sensor. Comparison is made by the sensor every five seconds.

If the current behaviour profile does not fall within the accepted behaviour pattern defined by the user historical profile, the comparator provides the transmitter with the following information that is then sent to the core agent: user identifier, invalid behaviour type and corresponding invalid behaviour type data. For example, if the invalid behaviour type were *unauthorised directory access* then the invalid behaviour type data would be the name of the directory attempting to be accessed. When invalid behaviour occurs, several courses of action are available. The choice depends on the severity of the invalid behaviour. The options can be a combination of:

i. Warn the system administrator.
ii. Warn the end user.
iii. Kill the specific application that has caused invalid behaviour.
iv. Prevent the end user from running any further applications.
v. Prevent the end user from running any applications.

Cases (ii-v) can be achieved locally at the client workstation. Case (i) is achieved by the user agent informing the core agent and the core agent informing the systems administrator. The user agent terminates when a user logs off.

Within a period of a day, users were allowed to log on and log off and work freely. However, each user was also given instructions to:

- Read the contents of any directory for which permission was not granted.
- Read, write or execute any file from any directory for which permission was not granted.
- Detect and shut down the user agent.
- Report changes in the performance of the computer network.

With the server and core agent operating as described above, we observed that a user agent was successful in detecting attempted breeches of security using both methods of statistical anomaly and rules bases, The range of invalid behaviour was such that each course of action was exercised. However, the software agent system cannot detect breeches of security when a valid use of an application transmits data to the server and the data can cause breeches of security on the server.

As part of the experiment, we killed off the core agent process. The implications of this were that new users posed a high security risk because neither the user's historical profile or systems administrator's behaviour rules were downloaded to the user agent. The detection of invalid behaviour from existing users continued unchanged but without reports being made to systems administrators of their invalid behaviour.

The user agent process was automatically given administrator access rights when the user logged in and when the user agent process was created. This meant that although a user could detect a user agent it could not be shut down.

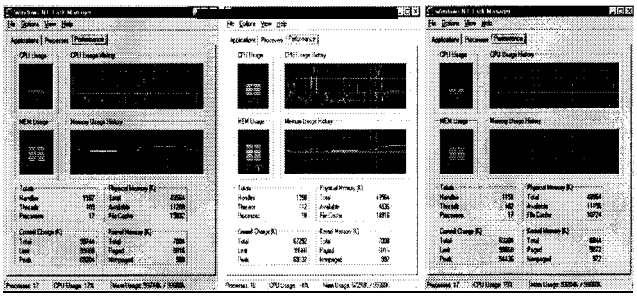

Figure 4 NT task manager

Some users did report a perceived drop in performance. Figure 4 shows the status of a workstation before, during and after a user agent's operation. During the operation of a user agent the available memory decrease is small but the CPU Usage increases from approximately 19% to 40%. The core agent was written in Java, a machine-specific executable *.exe* tile generated. The user agent was a combination of Java classes and C++ dynamic link libraries, which were used in the sensor to retrieve the details of a user's identifier and each application's name and process identifier. The Java Virtual Machine (JVM) environment was deployed to run the user agent software. The increase in CPU usage was attributable to the running of JVM, which had a run-time priority, set to low.

5 Discussion

To date we have only experimented with a network topology of up to 10 clients. We believe that further increases will not adversely affect the performance of the system in responding to security breeches. This is because the responsibility for security detection resides primarily with a user agent and each user agent operates independently from every other user agent. Only if many users logged in simultaneously and hence the core agent downloaded many user historical profiles and rules at the same time, would security be at risk because of the delay between responding to the first user and the last.

We do not regard as critical the speed with which informa-

tion about invalid behaviour is transmitted from the user agent to the core agent. This is because it is the user agent that is taking the action against any security breech. The information that sent to the core agent is a log of events to be processed by systems administrators at their convenience. Thus adding extra clients will not place a significant additional burden on the performance of transaction processing between core agent and user agent.

WC chose the period of one month over any shorter period to analyse a user's typical behaviour pattern believing this period would provide a representative profile of user behaviour. We could have taken a longer period than a month. This may have provided a more accurate picture of behaviour. However to do so would have meant maintaining an audit-log file of considerable size. For example, the audit-log tile for 10 users over a one-month period was about 200MB. This may jeopardise the performance of the core agent and other processes running on the server. In addition, it would increase the size of each user historical profile thus increasing the time taken to download this profile to the user agent. Since a user agent prevents a user from running any applications until the user historical profile and rule set have arrived from the core agent, the size of these files can affect the length of time a user must wait to login before they can commence working. There must be a balance between downloading files containing sufficient behaviour information to prove effective in security detection and prohibitive login times.

The set of user historical profiles also can cause space problems if the average profile is 500KB and there are a large number of users. To save space one approach is to create groups of users and to generate historical profiles of group behaviour. Individual user behaviour can go into user historical profiles. When a user logs in both the user's group behaviour profile and user historical profile are downloaded to the user agent and used by the comparator. However the downloading of two profiles rather than one may increase the download time.

Several approaches can be used for adapting a user's historical profiles. One approach is that a system administrator updates the user historical profile after judging that the illegal behaviour that triggered a warning from the core agent was in fact permissible behaviour for that user. Another approach is that user historical profiles are re-created every month by merging the previous month's behaviour pattern with the current month's behaviour pattern derived from statistical analysis of the current month's audit log file.

Currently the security breeches are detected by comparing a behavioural event with valid behaviour after the event has occurred. This has the limitation that some damage may have been caused in the time it takes the user agent to detect the invalid behaviour. For example, if a user starts Word, then starts a file manager and then runs a command line shell in order to read other users tiles, it is only after user files have been read that a security breech is reported.

One approach to managing this is for the user agent to take over the scheduling of processes from the operating but this is not practical. An alternative approach is to predict a set of events from a given series. This is a focus of our current work. For example the user agent can predict that if the first two steps occurred then there is a possibility that the third step will occur, thus issuing a warning to the user that it can execute the third step but he or she is being monitored.

The rules governing illegal behaviour are reviewed monthly at the same time as the audit-log tiles are analysed. Over time, we would expect that the rate of change to the rules would decrease.

Since the responsibility for security detection resides with a use agent, once the user historical profile and systems administrator's behaviour rules are downloaded to the user end agent, system administrators can take the server down for a maintenance or any other reason without jeopardising system security.

6 Conclusion

We built security enhancement software system using software agents, in which a core software agent resides on one server in a Windows NT network system and user end software agents reside in each user workstation. The significance of our approach is that the user agent at the client workstation takes all decisions and actions about invalid user behaviour. This means that even when the core agent process is no longer operating security breeches at the client workstation can still be detected and an appropriate course of action taken.

7 References

[1] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The *Architecture of a Network Level Intrusion Detection System* Technical Report Dept. of Computer Science, University of New Mexico, New Mexico, August 1990.

[2] National Institutes of Health. Center for Information Technology.http://www.alw.nih.gov/Security/securityprog.html#commercial, October 1998.

[3] W.J. Buchanan. *Handbook of Data Communications and Networks*. Kluwer, 1998.

[4] Chris Herringshaw. *Detecting Attacks on Networks*. IEEE Computer Magazine, pages 16-l 7, December 1997.

[5] Debra Anderson. *Detecting Unusual Program Behavior Using the NIDES Statistical Component*. IDS Report SRI Project 2596, Contract Number 910097C (Trusted Information Systems) under F30602-91-C-0067 (Rome Labs), 1995.

[6] T. Lunt, H. Javitz, and A. Valdes, et *al. A Real-Time Intrusion Detection Expert System(IDES)*. SRI Project 6784, February 28 1992. SRI International Technical Report.

[7] Sandeep Kumar and Gene Spafford. *A Pattern Matching model for Misuse Intrusion Detection*. Proceedings of the 17th National Computer Security Conference, October 1994.

[8] Mark Crosbie and Gene Spafford. *Active Defence of a Computer System using Autonomous Agents*. COAST Group, Dept. of Computer Science, Prudue University, Technical Report (95-008):2–3, February 15 1995.

[9] The Computer Misuse Detection System. http://www.cmds.net/, 1998.

[10] M. Wooldrige and N. Jennings. Intelligent Agents: Theory and Practice. 1995.

[11] M. Sobirey, B. Richter, and H. Konig. *The Intrusion Detection System AID*. Architecture, and Experiments in Automated Audit Analysis. Proceedings of the IFIP TC6/TC11, pages 278–290, September 1996.

[12] Diego Zamboni, Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernadez, David Isaco, and Eugene Spaord. *An Architecture for Intrusion Detection using Autonomous Agents*. COAST Technical Report 98/05, COAST Laboratory, Purdue University, Purdue University, West Lafayette, IN 47907-1398, June 11 1998.

[13] Donald P. Gaver and Gerald L. Thompson. Programming and Probability Models in Operations Research. Brooks Cole Publishing Company, Monterey, California, USA, 1987.