

A COMPREHENSIVE SIMULATION PLATFORM FOR INTRUSION DETECTION IN DISTRIBUTED SYSTEMS

K. Mantha, R. Chinchani & S. Upadhyaya
Department of Computer Science & Engineering
State University of New York at Buffalo
Buffalo, New York 14260
shambhu@cse.buffalo.edu

K. Kwiat
Air Force Research Laboratory
525 Brooks Road
Rome, New York 13441
kwiatk@rl.af.mil

KEYWORDS

Computer security, distributed systems, intrusion detection, simulation

ABSTRACT

This paper¹ describes the simulation of an attack recognition system in a distributed environment. The underlying technique of attack recognition is based on assertion checking. An auxiliary process called watchdog queries the users for a scope-file, from which an assertable plan called Sprint plan is generated. The sprint plan consists of carefully derived assertions, which forms the basis for attack monitoring. Two environments are simulated for the purpose of testing and evaluation of the intrusion detection system. First, a general academic environment with limited security restrictions is simulated. Second, a virtual banking environment with stringent security requirements is simulated. Different attack scenarios are simulated for the purpose of testing the recognition system.

1 INTRODUCTION

As today's commercial applications are becoming more and more distributed, they also become vulnerable to attacks across the network. This difficulty raises the need for concurrent intrusion/anomaly detection. Though a variety of techniques exist, the detection latency of these systems remains high, since most of them rely on audit trail analysis as the base-line approach. Unfortunately intrusion detection schemes based on audit trail analysis do not offer much in terms of damage containment, as they are passive, after-the-fact solutions [1].

¹The work was supported, in part, by the 1999 Summer Research Extension Program, AFOSR Contract F49629-93-C-0063

This paper describes the simulation of an on-line attack recognition system in a distributed environment. The underlying technique is described in detail in a previous paper [2]. In this scheme, prior to starting a session on a computer an auxiliary process called a watchdog first queries the users for a scope-file and then generates a table called Sprint plan. The sprint plan is composed of carefully derived assertions that can be used as a basis for concurrent monitoring of user commands. The plan is general enough to allow a normal user to perform his task without much interference from the watchdog and is specific enough to detect intrusions both external and internal. The focus of current paper is the simulation of two environments for the purpose of testing and evaluating this intrusion detection system. First, a general academic environment with limited security restrictions is simulated. Second, a virtual banking environment with stringent security requirements is simulated. Different attack scenarios, both internal abuse and external attacks are simulated and tested.

Some background and related work are given in Section 2. An overview of the concurrent intrusion detection system is given in Section 3. Section 4 describes the implementation of the watchdog monitor. The simulation of the test environments is described in Section 5. Experiments and results are given in Section 6. The paper concludes with a brief discussion.

2 BACKGROUND

Intrusion detection is a critical phase of information survivability. A model for intrusion detection was first given by Denning [3], which uses audit trail analysis as a basis of detection. The existing intrusion detections systems (IDS) are frequently classified into misuse and anomaly detection models [4]. The anomaly detection approach is based on the idea that an attack on a system will be different from the normal activity and an intruder

will exhibit a pattern of behavior different from the normal user [5]. In the misuse detection approach, the IDS watches for indications of specific, precisely representable activities of system abuse. The IDS includes a collection of known intrusion signatures, which are encapsulations of identifying characteristics of specific intrusion techniques. If a new attack comes in, the system is most likely to fail to detect it. USTAT is a Unix based real-time intrusion detection tool developed by Ilgun et al. [6]. It uses state transition analysis where penetration is identified as a sequence of state changes from a known initial state to a final compromised state. Several real-time network-monitoring tools have been developed over the past years. NADIR [7] is a Network Anomaly Detector and Intrusion Report tool. DIDS [8] is Distributed Intrusion Detection System, which looks and correlates multiple machine connections to the initial login. The project called EMERALD (Event Monitoring Enabling Response to Anomalous Live Disturbances) [9] aims at developing a distributed monitoring scheme, which uses a combination of signature engine and profiler engine within the monitor. The research team at Purdue is developing an adaptive network monitoring technique using autonomous agents [10].

3 OVERVIEW of CIDS

Our concurrent intrusion detection system (CIDS) uses verifiable assertions similar to the notion of control flow checking in fault tolerance [11], [12]. In this scheme, the user starts a session on a computer in a standard way by logging in. The system's watchdog then queries the user for a session-scope. This is the summary of the intended system usage in that particular session. Once the scope-file is submitted, the user is allowed to continue with his session. Before the user starts his session the watchdog does the monitoring of the user. This involves the translation of the scope-file into a sprint (Signature Powered Reasonable Instruction Table) plan. This gives a mechanism for monitoring the user behavior. When the user is in session, the watchdog monitors the user commands and checks if the command is the one he originally intended to execute. Any significant deviation from the plan is an indication of potential intrusion. Figure 1 shows the basic block diagram of the CIDS.

To implement this scheme in a distributed computing environment, a watchdog process is set up for each user on a given node. However, the process remains dormant until a user starts a session on a node. These watchdogs are essentially instances of the same process, monitoring the various sessions. They remain restricted to local nodes, but once operational interact with a master watchdog which is responsible for the coordinated dis-

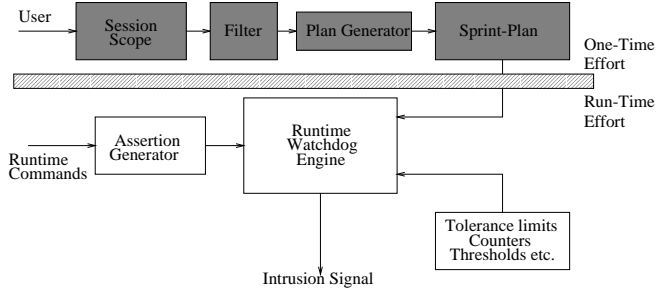


Figure 1. Flow Diagram of CIDS

tributed system monitoring. Figure 2 shows the overall architecture for the network of computers consisting N nodes and a file server.

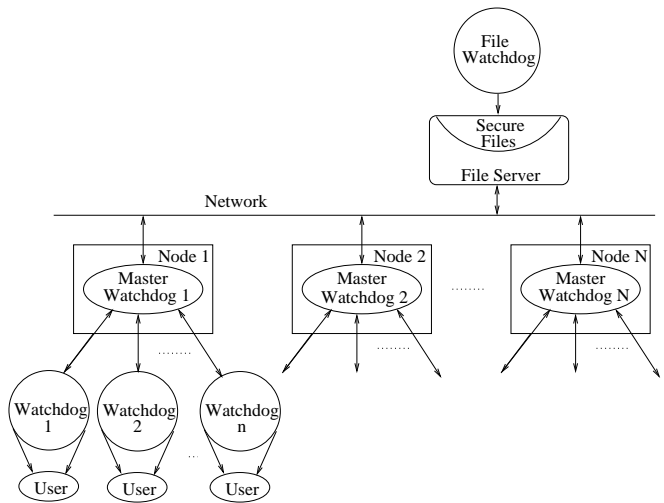


Figure 2. Architecture of Distributed CIDS

Each local area network has a separate watchdog called a file watchdog. The function of the file watchdog is to monitor access to secure files on the file server as shown in the figure.

4 IMPLEMENTATION OF CIDS

The session-scope can be fed into the system in a variety of ways. We use a graphical user interface (GUI) to simplify the process of the user input. With the help of the GUI a few simple checks are done while logging into the system. Figure 3 shows the flowchart of the steps taken by the watchdog when the user logs in. If the login is valid, the watchdog queries the user about the application the user is going to work in that particular session. Based on the application a preselected list of inputs containing the system resources available for the user is provided in a GUI, where the user can select the scope of

the session. The watchdog queries the user about his multiple login intent. If the user wishes to open multiple sessions, a list of all the hosts a user can connect to in the network is generated. The scope file thus obtained is given to a converter, which is built into the watchdog. The converter converts the scope-file into a sprint plan consisting of verifiable assertions. A verifiable assertion is a quadruple of the form: $(Subject, Action, Object, Period)$ where *subject* is a user (along with additional identifiers such as terminal ID, IP address etc.), *action* is an operation performed by the subject such as login, logout, read, execute and *object* is a receptor of actions such as files, programs, messages, records, terminals, printers etc. A temporal characteristic called *period* signifies the time interval for the usage of a given user command. A formatter formats this spring plan into a format that can be used for comparison.

It is possible that a user's scope exceeds the pre-selected system resources input list and so the watchdog provides the user with a text area, where the user can type his intent. This text is sent to an intelligent and learning software agent to convert it into the sprint plan. The architecture of this agent is shown in Figure 4.

The watchdog sends the user-defined text to the software agent using the watchdog/agent interface. This is then passed on to a parser, which parses the text looking for known keywords. These are then sent to the execution module. The job of the execution module is to schedule the inputs coming from individual watchdogs as there is one software agent per master watchdog even though there is one watchdog per user, to reduce the overhead on the system. The execution module then looks into the agent database to generate the appropriate mapping for the keywords. If it is not available in the database then it tries to do its job by communicating with other agents on the network doing similar work. The communication between the agents is achieved using KQML (Knowledge Query and Manipulation Language), a protocol that extends these agents to share their knowledge and work towards cooperative problem solving. This is achieved using the agent/agent interface. Once the mapping is done, it is formatted into the sprint plan.

Once the one-time monitoring effort is complete i.e., the sprint plan is generated, the user is allowed to proceed with his normal operation. Every user activity on the system is converted to an atomic operation by the watchdog's preprocessor. The output of the preprocessor is similar to the sprint plan, and is used by the watchdog for comparison with the actual sprint plan. Site-specific details, if any, are also given to the comparator. Any violation is reported to the master watchdog. The architecture of the monitor/comparator unit is shown in

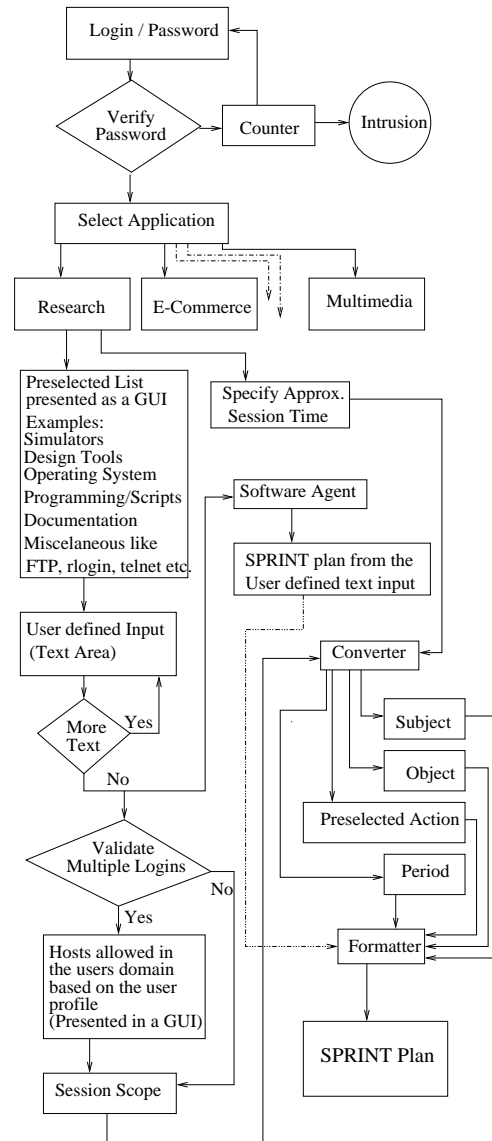


Figure 3. Flowchart of Sprint Plan Generation

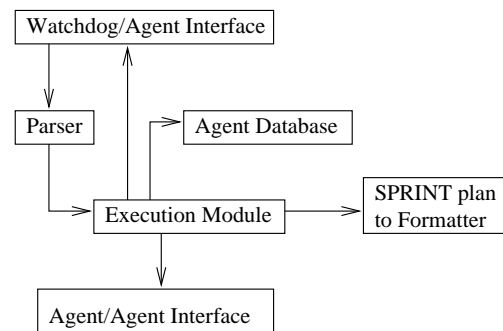


Figure 4. Software Agent Architecture

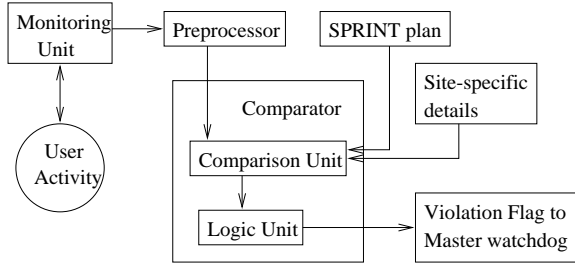


Figure 5. Monitor/Comparator Unit

Figure 5.

The comparator consists of a comparison unit and a logic unit. The comparison unit does all the comparisons and if there is a mismatch then it is passed to the logical unit, which determines the violation flag level, by which the counters are to be raised. For instance, if the user selects a Unix system and types “*I want to work on my research paper*”, then the software agent resolves the text into a set of atomic operations. In this case, these atomic operations can be treated as commands that are typed at the Unix prompt. If the agent knows that this particular user uses LaTeX software, the output from the agent will be *latex#bibtex#slitex#ispell#text-editor*. This is based on the fact that the user will use a text-editor to type in the contents of the paper, and later will invoke the LaTeX software by typing the command *latex* at the Unix prompt. If he uses *latex* then he probably will use *bibtex* and so on. The # here is just a separator between different commands.

5 TEST ENVIRONMENTS

We chose two distinctive test environments for simulation. The first one is a student/faculty user environment in a university setting with limited security constraints. The second one is a virtual banking environment with stringent security requirements.

5.1 Simulating a University Environment

The basic architecture is client-server based. Such a setup allows us to derive some test cases from the published descriptions of well known attacks and in developing site-specific test cases based on the security policy. It also helps us to consider both sequential and concurrent intrusions. In a sequential intrusion, a single person issues a single sequence of commands from a single terminal or a workstation window. In concurrent intrusion, one or more intruders issue sequences of commands from several terminals, computers or windows. The command sequences work cooperatively to carry out an at-

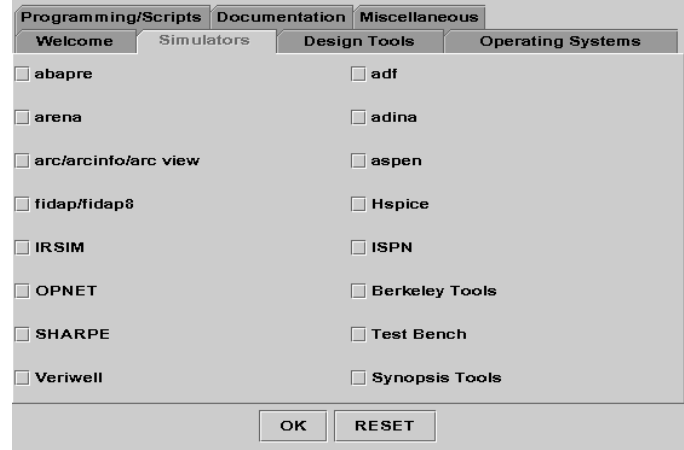


Figure 6. Choice of Simulators on a System

tack. For example an intruder can open multiple windows on a workstation and connect to a target computer from each window and try to distribute the suspicious behavior among them. The platform allows us to simulate basic sessions such as telnet, ftp etc. Synchronization can be achieved which lets us specify a fixed execution order of events.

When the student/faculty user logs in with a userid/password submission, password verification is done first. If the user is authenticated to login he will be provided with a series of GUI windows to specify the scope of the session. The user selects the application he is going to work on. If, say, the user selects Research as the application, the user is provided with a preselected input list containing various categories such as simulators, design tools, operating systems, programming languages, scripts, documentation and miscellaneous items such as ftp, rlogin etc. Figure 6 shows the screen image of the GUI illustrating the simulator choices.

The user just needs to check the tasks he intends to perform. Once this is done the watchdog queries the user if he intends to perform any other activities that are not present in the predetermined list. The user is also queried if he intends to open multiple sessions.

The various components of the sprint plan (not shown) are combined together by a formatter to obtain the final sprint plan. Figure 7 shows a run-time monitoring setup for a 1-user, 2-hosts system on a single server.

The sprint plan generated for the user is stored at a secure location on the server. As soon as the user logs in to a host, the watchdog checks to see if there is a sprint plan already existing for the user, if there is none, it generates a new one. If a sprint plan already exists, the user is allowed to proceed with his normal activity. The watchdog continuously monitors the user and compares

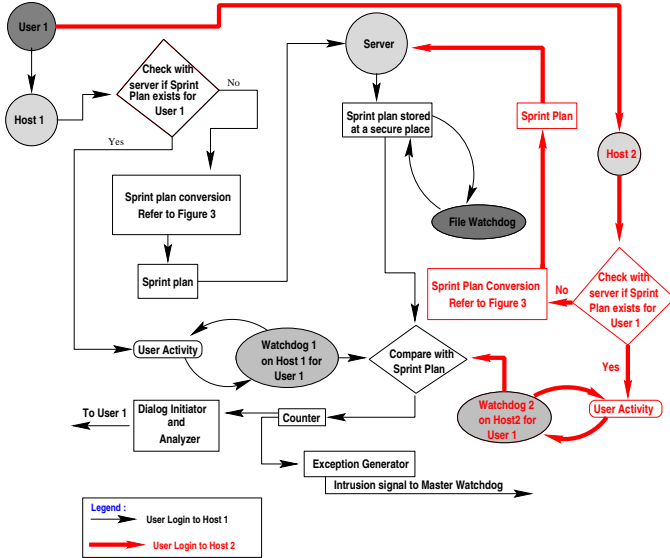


Figure 7. Run-time Monitoring Setup

it with the sprint plan.

5.2 Virtual Banking Test Environment

The second test platform is a virtual banking environment with stringent security requirements. The main focus here is on the employees in the bank who are the users of the banking software. Misuse intrusions can be tested in detail here. The employees in a bank might have access to sensitive data and there is always a possibility of misusing privileges. The environment supports multiple banks with multiple accounts, with multiple access to all the accounts. The virtual banking system can be accessed by a wide variety of employees starting from the president of the bank to the teller in a bank. A session-scope is provided by the employees who login to the system with an authenticated userid and password with the help of a GUI. Once the user submits his scope file, the watchdog converts it into a sprint plan. Once this conversion is complete, the user gets the access to the bank system.

The employees can work in different areas of the banking system such as bank operations, business banking, credit/lending division, e-commerce, investment banking, monitoring etc. If the user is working in the bank operations division then some of the features provided to the employee are creating new accounts, opening existing accounts etc. It also supports a variety of accounts such as checking, savings, loan, mortgage, etc. The system allows the user to do things such as profitability analysis, generate reports etc. The system contains two databases, which the user can access. A transfer tool is provided for

transferring balances for one account to another and from one database to another.

The watchdog continuously monitors the user activity in the system. The file watchdog is built into the systems database, which monitors the database accesses and communicates with the master watchdog to report any illegal transactions or access violations. The system is written in Java [13] and uses SQL queries to query the database. The database is custom made and is also implemented in Java. JDBC has been used for the connectivity between the bank system and the bank database. All the GUI components are light-weight, built using JFC Swing [14].

6 EXPERIMENTAL RESULTS

We report the results of our simulation on the university environment only. The metrics used are detection coverage and performance.

There is usually no simple procedure to identify appropriate test cases for an intrusion detection system. A variety of intrusion scenarios are considered based on some common practices of system usage. These scenarios are grouped into four categories, viz., one-user without multiple logins, one-user with multiple logins, multiple users without multiple logins and multiple users with multiple logins. Two set of experiments are performed in each of these categories, first with the worst case, where a user selects all the entities provided in the session-scope GUI by the watchdog and the second where a user selects only a few entities. The tests are performed by treating the logins as four different cases, with up to two users at a given time. The first case is where both logins are legitimate. In the second case, the first login is from a legitimate user and the second login is from an intruder. In the third case, the first login is from the intruder and the second login is from the user and finally the fourth case where both logins are from intruders.

Some of the selections of the intrusive activity simulated are transferring the `/etc/passwd` file from one host to another, password-cracking by comparing the entries in the `/etc/passwd` file to entries in another file, using a dictionary file for the same, exploiting the vulnerabilities such as `rdist`. The system is able to detect all the intrusive activities and terminate the connection for the logins of intrusive users except in the worst case scenarios of one-user with multiple logins and multiple users with multiple logins, where a small number of intrusive activities was not detected. The system has also generated a few false positives, flagging an intrusion when normal user activity is taking place. This happens when the user selects only a few entities from the session-scope.

Since Java is used for implementation, moderate impact on system performance is expected. When new con-

nections are made or more users login, the system load increases. However, this increase is only marginal because there is no need to maintain any large data structures for each user or connection. The main server on which CIDS is running is a Sun Ultra Enterprise 450 Model 4400 and the clients are Sun Ultra 5's running Solaris 2.7.

A normal user in a university environment is assumed to have about six to eight processes running on the system at a given time. There is one watchdog dedicated for each user which makes it one more process per user on the system. This watchdog process does not use many run-time resources and hence may not become an overhead to the system.

In order to quantify the overhead of the CIDS, we eliminated all unrelated activities in the test environment, started the CIDS and allowed the users to login. We analyzed the average load per minute (no. of jobs in the run queue on Unix) and the storage overhead in kB against the number of users on the system. At this particular stage of implementation without much optimization, the operation is very stable for about 15 users. The load on the system tends to increase as the number of monitored users increases beyond 15. The storage overhead (325 kB for a single user) increases at a constant rate with the number of users. When the session-scope is large, the watchdog maps it to a huge sprint plan. The storage used by the CIDS in our study corresponds to the worst case scenario where a user selects all the entities from the session-scope provided by the watchdog in a GUI.

7 CONCLUSION

The concurrent intrusion detection prototype described in this paper is in its preliminary stage and the experiments reported are very basic. Our simulation shows that on-line intrusion detection using assertion checking is feasible, that is, low performance overhead and good detection coverage. This approach is an alternative to conventional audit trail analysis based intrusion detection schemes.

More detailed experiments with complex intrusion scenarios are planned. This requires further enhancements to the sprint plan generation and consideration of structural and temporal sequence checking. Similar experiments will also be conducted on the virtual banking application.

References

- [1] J. Feldman, J. Giordano, and J. Palmer, "Information survivability at Rome laboratory," *1997 IEEE Information Survivability Workshop*, 1997.
- [2] S. Upadhyaya and K. Kwiat, "A distributed concurrent intrusion detection scheme based on assertions," *SCS Int. Symp. on Perf. Eval. of Comput. and Telecom. Systems*, pp. 369–376, July 1999.
- [3] D. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 222–232, February 1987.
- [4] S. Kumar and E. Spafford, "A pattern matching model for misuse intrusion detection," *Proceedings of the 17th National Computer Security Conf.*, pp. 11–21, October 1994.
- [5] H. Javitz and A. Valdes, "The sri ides statistical anomaly detector," *Proc., IEEE Symp. on Research in Security and Privacy*, pp. 316–371, May 1991.
- [6] K. Ilgun, R. Kemmerer, and P. Porras, "State transition analysis: A rule-based intrusion detection approach," *IEEE Trans. on Software Eng.*, vol. 21, pp. 181–199, March 1995.
- [7] J. Hochberg, K. Jackson, C. Stallings, J. McClary, D. DuBois, and J. Ford, "NADIR: An automated system for detecting network intrusions and misuse," *Computers & Security*, vol. 12, pp. 253–248, May 1993.
- [8] S. Snapp, S. Smaha, T. Grance, and D. Teal, "The DIDS Distributed intrusion detection system prototype," *USENIX, 1992 Technical Conference*, pp. 227–233, June 1992.
- [9] P. Porras and P. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," *National Information Systems Security Conf.*, pp. 353–365, Oct. 1997.
- [10] J. Balasubramaniyan, J. Omar, G. Fernandez, E. Spafford, and D. Zamboni, "An architecture for intrusion detection using autonomous agents," *Department of Computer Sciences, Purdue University, Coast TR 98-05*, 1998.
- [11] M. Namjoo, "Techniques for concurrent testing of VLSI processor operation," *Proc. International Test Conference*, pp. 461–468, November 1982.
- [12] S. Upadhyaya and B. Ramamurthy, "Concurrent process monitoring with no reference signatures," *IEEE Transactions on Computers*, vol. 43, pp. 475–480, Apr. 1994.
- [13] P. Deitel and H. Deitel, *Java How to Program*. Prentice Hall, 1999.
- [14] S. Pantham, *Pure JFC Swing*. SAMS, 1999.