

# A DISTRIBUTED CONCURRENT INTRUSION DETECTION SCHEME BASED ON ASSERTIONS

Shambhu J. Upadhyaya  
Department of Computer Science & Engineering  
State University of New York at Buffalo  
Buffalo, New York 14260  
shambhu@eng.buffalo.edu

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

## KEYWORDS

Assertions, Computer security, distributed systems, information assurance, intrusion detection

## ABSTRACT

*This paper<sup>1</sup> presents a new technique for intrusion detection based on concurrent monitoring of user operations. In this scheme, prior to starting a session on a computer, an auxiliary process called watchdog first queries users for a scope file and then generates a table called a 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 or system administrator and is specific enough to detect intrusions, both external and internal. A distributed watchdog process architecture based on the notion of verifiable assertions is presented. This scheme is a significant enhancement over the traditional approaches that rely on audit trail analysis in that the intrusion detection latency could be much shorter.*

## 1 INTRODUCTION

Information survivability in the wake of abuse, misuse and malicious attacks is an important issue in today's distributed systems. It is necessary to develop countermeasures to deal with the attacks or intrusions since some of the attacks may succeed despite vigorous information security measures. It is almost impossible to close all security loopholes and guard against malicious break-ins as well as abuse of systems by legitimate users [1]. One of

the countermeasures is to detect intrusions immediately and initiate recovery procedures to undo the damage.

A variety of intrusion detection techniques and tools exist in the computer security community. Though these techniques follow different approaches for intrusion detection, audit trail analysis has been used as the last line of defense [1]. In these methods, the user behavior is monitored for certain patterns of abuse by looking at the audit data. Unfortunately, intrusion detection schemes based on audit trail analysis do not offer much in terms of damage containment because these approaches are passive, after-the-fact solutions [2]. In order to contain the damage effectively it is essential to detect intrusions concurrently so that recovery and restoration of service can be expedited. The development of such a concurrent monitoring scheme is the main focus of this paper. The low detection latency is a significant feature of our technique that can make it highly supplemental to existing core anomaly detection algorithms.

In order to develop concurrent monitoring of any operation and flag an exception, one must have a flow graph of the intended operations. An analogy exists in the traditional fault tolerance area where concurrent error detection has been successfully implemented at instruction level using the control flow of user application programs [3]. However, intrusion detection is different from this because intrusions may not manifest as instruction-level control flow anomalies. Instead, one must look at a higher level such as the user command level where intrusions can be better monitored. However, control flow of user operations is not known *a priori*, making concurrent monitoring a difficult task. As an alternative, we devise a novel reference table to compare the user operations with.

The proposed technique of concurrent intrusion detection starts by first generating a plan for the potential user and then monitoring the user for any significant deviation from this plan. This plan is composed of verifiable

---

<sup>1</sup>The work was supported, in part, by the US AFOSR 1998 Summer Research Program

*assertions* that are automatically generated on the basis of a few system usage inputs provided at the beginning of a session. Once an assertable plan is generated, the user will be monitored to see how close the plan is maintained during a given session. Any significant deviations from the plan can be construed as an intrusion.

Our approach is based on sound principles that have been successfully used in concurrent monitoring of processor operation. In addition, the new approach offers several advantages. First of all, the technique is not based on audit trail analysis. Audit trails are generally huge and the analysis and filtering out of useful information is expensive [1]. The assertable plan can take into account user dynamics and any intended changes in the user behavior. Therefore, no special learning is needed as in rule-based systems. Both inside abuse and external intrusion can be minimized. Since our scheme does concurrent monitoring with low latency of detection, it can be combined with any recovery schemes for effective damage containment and restoration of service.

Some background and related work are given in Section 2. An overview of the methodology and an architecture of distributed concurrent intrusion detection appear in Section 3. A basic algorithm and some enhancements are given in The paper concludes with a discussion and future plans.

## 2 BACKGROUND

Intrusion detection which is concerned with the detection of unauthorized access of computer systems is a critical phase in information survivability [4]. A model for intrusion detection was first given by Denning [5] which uses audit trail analysis as a basis for detection. Lunt [1] provides a good survey of traditional intrusion detection techniques.

Feldman et al. [2] observe that the current intrusion detection technology focus is largely *after-the-fact*. What is needed is a proactive/predictive intrusion detection technology incorporating advances in pattern matching, user profile analysis, and intrusion signature recognition. The technique proposed in this paper supports the building of self-aware hardware and software systems that adapt, in real-time, to external and internal anomalies.

Clyde [6] suggests *keystroke* monitoring as a means to dynamic intrusion detection. This is a behavior pattern monitoring which can be done on-line unlike audit trail analysis which is essentially *after-the-fact*. The idea is that the intruder will have a different keystroke pattern compared to the legitimate user's. This technique has been discounted as impractical by Kumar and Spafford [7] due to the myriad ways of expressing the same at-

tack at the keystroke level. Aliasing has also been cited as a reason for the defeat of this technique. Command-line auditing has been tried on Unix systems by several researchers as a means to intrusion detection. This technique is ad hoc and is similar to keystroke monitoring. USTAT is a Unix-based real-time intrusion detection tool developed by Ilgun et al. [8]. It uses a state transition analysis where penetration is identified as a sequence of state changes from a known initial state to a final compromised state. The technique is audit trail-based. It works on the data collected by the C2 basic security module of SunOS and keeps track of only those critical actions that must occur for the successful completion of penetrations. The work of Kumar and Spafford [7] uses a novel pattern matching scheme for real-time intrusion detection. Their model provides the ability to specify partial orders and subsumes matching of sequences and regular expressions. Generality, portability and flexibility have been cited as the main benefits of this model.

Several real-time network monitoring systems have been developed over the past years. NADIR [9] is a network anomaly detector and intrusion report tool which was originally designed to accept audit logs from a Los Alamos network security controller running a homegrown version of Kerberos. A more powerful version of NADIR called UNICORN accepts audit logs from Cray Unix called UNICOS, Kerberos and common file systems and analyzes them and tries to detect intruders in real-time. DIDS [10] is a distributed intrusion detection system which looks at and correlates the connections on multiple machines to the initial login. More recently, the computer science lab at SRI International has undertaken a project called EMERALD (Event Monitoring Enabling Response to Anomalous Live Disturbances) [11]. This project is developing a distributed monitoring scheme which uses a combination of a signature engine and a profiler engine within the monitor for intrusion detection. The signature analysis is a process whereby an event stream is mapped against abstract representations of events that indicate undesirable activity.

The research group at Purdue is developing an adaptive network monitoring scheme using autonomous agents. Their approach [12] is distributed in the sense that one agent is used per node instead of a monolithic entity. The agent is somewhat similar to the monitor of Emerald [11]. This architecture also uses a hierarchical approach like Emerald.

## 3 METHODOLOGY OVERVIEW

Most of the schemes reviewed in the previous section are based on audit-trail analysis and pattern matching. Their real-time notion stems from the fact that they use either efficient pattern matching techniques or employ

distributed decision making to speed up detection. We approach the real-time intrusion detection problem from a different angle, namely, concurrent, on-line monitoring. This approach essentially does away with the passive audit trail analysis. The model and the simplifying assumptions are given next.

In our definition of a distributed system, we include a network of computers which service users on the basis of an account and a password. Further, we assume that the users on different machines have the same userID although the passwords could be distinct. This model precludes the monitoring of web surfing and anonymous ftp activities. No specific topology is assumed for the network. All the communications between nodes is by message passing and the network is assumed to be stable. This model makes our intrusion detection approach unique in that all intrusions are abstracted as happening through well-defined user sessions which are invoked through userID and password submission. This leads to the following observations.

1. The intrusions are of only two types: (a) external intrusions caused by logging into a legitimate user's account by way of cracking password or by joining an open session left logged on by a legitimate user; (b) internal misuse occurring by abusing the privileges or by forcing on others' accounts and manipulating system resources with a malicious intent.
2. The problem of intrusion detection simply transforms into monitoring the well-defined user sessions. No distinction exists between external or internal intrusions as far as the detection is concerned. Thus, external intrusions and internal abuse can be handled in a unified way.

We also assume that a user session on a node is of finite length. This facilitates consideration of temporal information for focused monitoring of user operations.

### 3.1 Definitions

**Definition 1** *An intrusion or anomaly is defined as any deviation from a "predetermined" operational behavior.*

The prior determination of legitimate operations or a sequence of operations is done based on certain criteria. This definition of intrusion includes such activities as masquerading, legitimate user penetration, and legitimate user leakage [5]. Other intrusions of the type Virus, Trojan horse, IP spoofing and denial-of-service are not explicitly considered here. However, the userID-password abstraction in our model may allow the detection of many cases of intrusion before leading up to anomalies such as denial-of-service.

**Definition 2** *A watchdog is a process that concurrently monitors user commands typed in from a keyboard or submitted in the form of a script or a macro.*

A watchdog process is spawned immediately following a userID and password submission and the creation of a user session. Only one watchdog process per userID is created. We assume that the watchdogs are run correctly on a given node. Subversion or disabling of the watchdog by intruders is possible but such activities can be detected by the system administrator.

**Definition 3** *Session-Scope is a file containing a list of intended activities submitted by a user at the beginning of a session.*

This file is a high level description of user operations in a prescribed format. Since only one monitor session is set up per user account, only one session-scope is accepted per user account even if multiple sessions are opened on a given user account. The session-scope, once submitted, is treated as a secure document and is not accessible to anyone. As will be seen later, having one session per userID over a given period of time facilitates easy monitoring for intrusion.

**Definition 4** *A verifiable assertion is a quadruple which associates a user with the intended operation over a given period of time. The format of the assertions is as follows.*

$$(subject, action, object, period) \quad (1)$$

where 'subject' is a user (along with additional IDs such as terminal identification, 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.

These verifiable assertions are generated in advance for each user event specified in the session-scope file. The temporal characteristic called *period* signifies the time interval for the usage of a given user command. For instance, this could be an absolute time interval.

**Definition 5** *A sprint-plan is a collection of verifiable assertions.*

This is a table automatically generated as a response to a user's session-scope file. The sprint-plan can also be viewed as a signed stream of commands that is generated, in no particular order, for the purpose of on-line monitoring.

### 3.2 Basic Principle

Our technique of intrusion detection using verifiable assertions is similar to the notion of control flow checking in fault tolerance [3], [13], [14]. In control flow checking, a preanalysis of the program is done to generate a control flow graph of the application. Signatures or assertions are embedded into the instruction stream at compile time to generate a reference graph. At runtime, the execution is monitored and at designated intervals, the runtime signatures are compared with predetermined signatures of the reference graph. Any discrepancy between the actual signature and the expected signature indicates an error. Both instruction level bit errors and control flow errors are detected by this scheme.

In our intrusion detection scheme, the user starts a session on a computer in a standard way, that is, by logging in. The system then queries the user for a session-scope. The user gives a summary of his intended system usage. An example of a session-scope is as follows. A user plans to work until 5 pm. He will be working on a research paper. He will perform email and web browsing. He will run some simulation on a particular server. He will clean up his mail folders and may do some other miscellaneous items if he has time before the end of the day. The purpose of such a plan file is to get a basic understanding of what the user's intention is. Enough flexibility must be built into the system so that the user does not constantly have to interact with the monitor. Once the scope file is submitted, the user is allowed to continue with his session. Meanwhile the system translates the scope file into a sprint-plan. When no ordering of events is possible on the activities of the user, the sprint-plan is simply a table of verifiable assertions. It has no control flow information as such.

The assertions give a mechanism for monitoring the user behavior. These assertions are generated automatically by reading the scope file and interpreting the user inputs properly. Verifiable assertions are a generalization of IDES's [15] specification of some very self-centric user characteristics so that such characteristics can be monitored on-line. An important component of our assertions is the subject field. The subject field is generated from the userID and other unique identifications such as the IP address of the workstation, tty number of the terminal being used etc. All such information will be coded into the subject field. For instance, a user may wish to open multiple login sessions. As long as such intent is expressed in the scope file, a more general subject coding can be done for this user in order to allow him to work from different terminals or set up multiple login sessions. As noted earlier, there is only one watchdog process per user per system even though multiple ses-

sions are opened.

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. Several techniques are used to minimize false alarms and to build robustness to the monitoring scheme. These details will be discussed later. The intrusion detection architecture is given next.

### 3.3 The Architecture

Our intrusion detection model is amenable to hierarchical monitoring where the lowest level of hierarchy is the user-session level monitoring. Hierarchical arrangement of watchdog monitors is highly effective in distributed systems as evident from other intrusion detection schemes such as Coast [12] and Emerald [11]. Figure 1 shows the overall architecture for a network of computers consisting of  $N$  nodes and a file server.

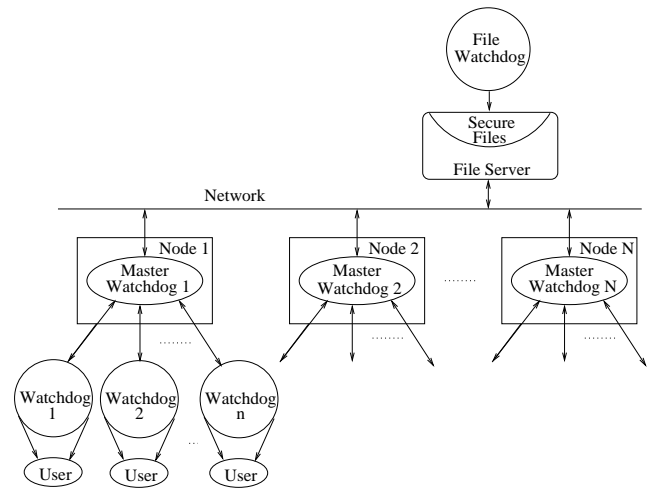


Figure 1. Overall Architecture of a Distributed Intrusion Detection Scheme

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 user sessions. They remain restricted to the local nodes, but once operational, interact with a master watchdog which is responsible for coordinating distributed system monitoring.

In addition to the user watchdogs and master watchdogs, each local network has a separate watchdog called a File Watchdog. The function of the file watchdog is to monitor accesses to secure files on the file server as shown in the figure. Such a separate watchdog with the

knowledge of the secure files and any privileged information will be more effective in detecting unauthorized disk accesses to secure data than incorporating the detection functions into the user watchdog. Since secure data may not reside on the disks of individual nodes, a central watchdog guarding the file server will be easier to implement than a distributed implementation. The file watchdog will interact with the master watchdog on the individual nodes to coordinate the dissemination of intrusion detection and to initiate recovery. The architecture of the individual user watchdog is shown in Figure 2.

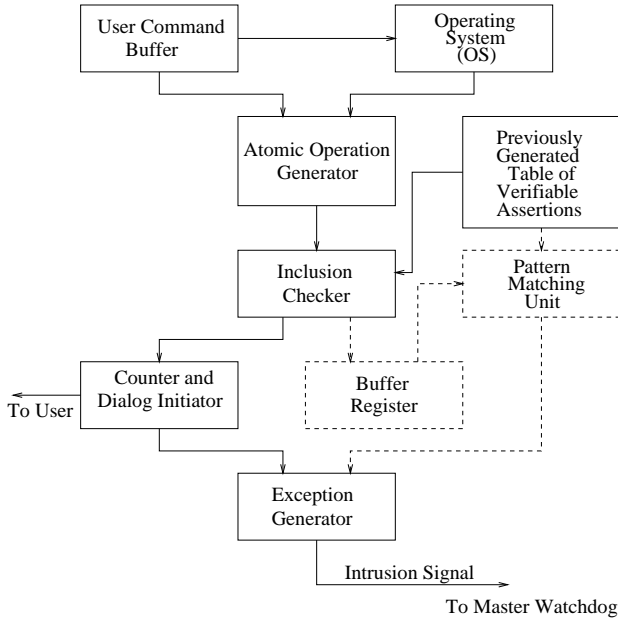


Figure 2. Block Schematic of the Watchdog Process

The watchdog process receives input from the User Command Buffer and/or from the OS. The Atomic Operation generator converts user command lines into an on-line assertion statement. The Inclusion Checker module verifies if the assertion statement generated on-line is in the set of previously generated verifiable assertions.

The watchdog has a dialog box which, with the help of a Counter, initiates a dialog with the user if the user is found to deviate significantly from the original plan. Based on the user response received at the Dialog Initiator, a decision is made to issue an exception to the master watchdog. An Intrusion Signal is issued by the master watchdog after carefully processing the monitored data.

## 4 A BASIC ALGORITHM AND ENHANCEMENTS

The concurrent intrusion detection scheme has two

phases of operation. The first phase is an initialization phase which solicits input from the user and generates a sprint-plan. We do not delve into this in this paper. The second phase is the runtime phase during which on-line monitoring is performed. The algorithm for the second phase is given below.

### 4.1 The Algorithm

A basic algorithm is given first. Certain tolerance limits are used to allow for user deviations from the intended plan. Appropriate counters are used to keep track of such deviations.

#### Algorithm:

1. Set *monitor\_rate*, *tolerance\_limit*, *counter*;
2. For all *user\_command\_line* do
3. Decode *user\_command\_line* into atomic operations;
4. If each atomic\_operation in *sprint\_plan* then
  - (a) no\_error, go to Step 3;
5. else
  - (a) If *subject\_ID\_violation* then
    - i. Set *intrusion\_signal*, exit;
  - (b) else
    - i. *counter++*; /\* increase count on permissible unplanned commands \*/
    - ii. If *counter*  $\geq$  *tolerance\_limit* then
      - A. If *provision\_for\_future\_changes* in *session\_scope* then
      - B. Reset *counter*, go to Step 3;
      - C. else
      - D. Issue message to user to update *session\_scope*;
      - E. If *user\_response* YES then
      - F. Compare new *session\_scope* with original *session\_scope*;
      - G. If *criteria* not met then /\* See explanation below \*/
      - H. Issue *intrusion\_signal*, exit;
      - I. else
      - J. Reset *counter*, go to Step 3;
      - K. else
      - L. Issue *Intrusion\_signal*, exit;
    - iii. else
      - A. go to Step 3;

One can choose to monitor every single user command or set a particular monitoring rate (Step 1) depending upon parameters like system load and specified security levels. The session-scope to be submitted at the beginning of a session need not be complete. The user can update his plan at a later time. The watchdog will query the user for an update based on a tolerance limit imposed on the amount of deviation from the original plan.

The user input line in the form of a singular command, command alias, macro or a script is converted into a set of atomic operations, irrespective of whether it is an interactive or a batch submission. These atomic operations will have the subject ID resolved based on runtime characteristics. If the atomic operations are found in the sprint-plan previously generated (Step 4), the session continues without interruption. If there is a mismatch, further examination is conducted to see if there is a subject ID violation (Step 5.a). If so, an intrusion is obvious and appropriate signals will be sent to the master watchdog. Otherwise, the mismatch is interpreted simply as a deviation from the original plan and the count on permissible unplanned commands is incremented (Step 5.b.i). If this count reaches a tolerance limit (Step 5.b.ii), a check is made to see if the user has made provisions in the session-scope file for future updates. If no such intent was specified in the session-scope file, a message is issued to the user to indicate that the original plan is too coarse and an update is in order. If the user responds affirmatively (Step 5.b.ii.E), he is expected to submit a revised session-scope. The revised session scope is examined for certain criteria. If the criteria are met, the counter is reset and the session continues without further query. The updated session-scope is translated to a new sprint-plan which will be used for the rest of the session or until the sprint-plan is updated again. If the criteria are not met or the user does not update his session-scope, an intrusion is flagged and appropriate action is initiated.

At Step 5.b.ii.F, a mere submission of an updated session-scope cannot be considered as a benign act. It is possible for a crafty intruder to perform only incremental changes or to include a very general plan in his updated session-scope and evade detection. Therefore, the updated session-scope must be examined for certain criteria. A comparison between the new file and the old file may reveal some of the user (intruder) intentions. One must also make sure that the updated scope file does indeed include the activities which triggered the counter overflow. Such checking will prevent intruders from continuing the intrusive activities by hiding the operations that triggered the counter overflow. Several such simple and effective criteria can be developed using reasonableness checks but this discussion is outside the scope of this paper.

## 4.2 Intrusion Scenarios and Enhancements

The algorithm presented above offers much flexibility and provision to the user to update his plan so that false alarms are minimized. The algorithm is also capable of detection of many intrusion scenarios. Consider the situation where two logins are made on the same account. Multiple logins is possible due to genuine use by the user or due to an intrusion while a legitimate user is in session. There are four distinct cases to consider.

- Case 1: Both logins are legitimate. This is the most common scenario. For instance, a user starts a session from his desktop and after a while goes to his laboratory and remotely logs on to his account from a terminal or workstation in the lab.
- Case 2: The first login is from the legitimate user and the second login is from an intruder.
- Case 3: The first login is from an intruder and the second login is from the legitimate user.
- Case 4: Both logins correspond to intrusions.

For all the four cases, irrespective of the number of logins, our model treats the usage of the account as a single session. So, there is only one session-scope file during the entire session.

### 4.2.1 Resolving the Cases

**Case 1.** In this case, the user is expected to include the intent of multiple logins in the session-scope file and hence, the session continues without interruption. If no request is made initially, remote logins or multiple logins from the same terminal is prohibited as a security measure.

**Case 2.** This case is handled easily if the user does not indicate the desire for multiple logins in his session-scope file. If the user admits multiple logins in his session-scope and the intruder is able to login from a prescribed terminal or workstation, the break-in succeeds. However, if the intruder is oblivious of the watchdog, he is likely to deviate from the activities given in the session-scope file and detection is imminent. Even if the intruder is aware of the monitor watchdog, his activity is likely to raise an exception since he has no knowledge of what is contained in the original session-scope file. Note that only the originator of the session can update the session-scope file.

There is some inherent security built into the algorithm if Case 2 occurs. If an intrusion occurs while the legitimate user is in session, the user will be contacted by the watchdog whenever the counter overflows (Step

5.b.ii). If the user believes that he has stayed within the session-scope for most part, he can choose not to update his session-scope. Too many requests from the watchdog to update session-scope is a clear indication that someone else has intruded the system.

**Case 3.** Case 3 occurs when the masquerader who logs in first admits multiple logins in his session-scope file. If the masquerader does not allow multiple logins to begin with, the legitimate user will be denied access when starting system usage. He is then expected to contact the system administrator. This will eventually result in intrusion detection. On the other hand, if multiple logins are admitted and if the legitimate user does not log in during the intruder's session, the intrusion continues to be in place and there is no way of detecting this aliasing problem using our algorithm. In such cases, we need to rely on other techniques such as recognition of unusual patterns in the user behavior. In the case where the intruder admits multiple logins and the legitimate user logs in while intrusion is in progress, the absence of a query from the watchdog to the user for a session-scope file may raise suspicion. The user, cognizant of the watchdog monitoring, is made to believe that someone is using his account before he has logged on or that the watchdog monitor itself is turned off. Even if the user takes no action (or say, is not cognizant of the watchdog), the continuation of his access will have a high probability of resulting in a significant deviation from what is in the masquerader's session-scope. The algorithm has the capability to capture such deviations.

**Case 4.** Case 4 can occur if the intruder himself initiates multiple logins or the second intrusion occurs from a different individual while the first intrusion is in progress. We assume that the probability of either of the events is small. However, when this happens, the latter scenario is identical to Case 3 if the second intruder is viewed as a user rather than as an intruder. In this case, the interactions of the events generated by the intruder and the user will eventually result in the detection. If the former situation occurs, the detection is not likely to happen by our algorithm and we need to rely on other techniques such as recognition of unusual patterns in the user behavior.

#### 4.2.2 Enhancements

Several refinements to the basic algorithm are possible to enhance the robustness of intrusion detection.

**Monitoring Sequence of Operations.** In Step 5.b.ii.F, certain criteria are used for discriminating intrusion from normal use. An enhancement can be added by buffering the on-line assertions and comparing sequences of assertions with some predetermined patterns indicative of possible abuse. This enhancement is shown in

Figure 2 by dotted lines. Also, one could look at the frequency of certain operations or other temporal characteristics of system usage to enhance the detection.

**Two Level Counters.** Another enhancement is possible in Step 5.b.ii.B. In this step, currently, the counter is simply reset to allow for anticipated deviations when the tolerance limit reaches. If there are significant deviations, the counter may reach the limit and get reset a few times without further scrutiny. When counter reset occurs repeatedly, a second counter can be used to permit only a certain number of counter overflows. This hierarchy of counters will add further security to the detection mechanism.

## 5 DISCUSSION

In this paper, we have presented a new approach to intrusion detection using verifiable assertions. We have developed this scheme by leveraging some of the successful concepts from the fault tolerance literature. The main feature of our technique is that detection occurs concurrently with user operations and in real-time. The low latency detection of our technique can potentially speed up the recovery of affected systems. This is a significant benefit compared to the schemes based on audit trail analysis.

We have given a basic architecture and an algorithm for intrusion detection. Several enhancements to the basic scheme are also presented. The technique is flexible in that changes or updates to the intended plan can be made easily. Also, every time a fresh session is started, a new set of verifiable assertions is generated. The finite length of a given user session helps to keep the sprint-plan to a small and bounded set.

The proposed technique is more effective in detection when intrusion occurs into logged on user accounts. This is because the intruder's operations are likely to result in large deviations from the intended session-scope of the user. Therefore, deactivating user accounts that are inactive for certain length of time will increase the probability of intrusion occurring in active user accounts where they can be detected. In the case where intrusion occurs when the (active) user is not logged on, the intruder routinely gets queried for a session-scope. This can be viewed as a deterrent because the idea that the system usage is monitored on-line may turn away 'casual' intruders.

The watchdog process which performs user session-level monitoring is the core module of the distributed concurrent intrusion detection scheme described in this paper. Our scheme is not intended to be a replacement to other intrusion detection tools such as pattern matching and rule-based systems built to work on audit trail data.

The concurrent monitoring watchdog described here can be used in conjunction with other distributed intrusion detection schemes to provide a higher detection resolution. For instance, the watchdog can be added as a third party security module in Emerald's monitor [11].

The intent of this paper is to present the concept of the new concurrent intrusion monitoring scheme and its technical details. Issues related to modern networks and abuse of network servers will be addressed in future versions. Our basic scheme will be first simulated/implemented in a distributed computing environment in order to study the performance implications.

Users of applications which work under controlled environments such as bank teller usage or other highly regular and restricted environment are expected to know in advance things like what programs they will execute, what files they will access, create, or destroy, what time they will log off, and whether they will require multiple simultaneous sessions etc. Such systems can benefit immensely from our technique. Even in less stricter environments, a user generally has a plan for his computing tasks on a given day. Therefore, the generation of the session-scope can be viewed as a simple routine if supported by elegant front-ends for the user input. We also have plans to automate the generation of the assertions using formal techniques such as accounting checks or reasonableness checks [16].

## References

- [1] T. Lunt, "A survey of intrusion detection techniques," *Computers and Security*, vol. 12, pp. 405–418, 1993.
- [2] J. Feldman, J. Giordano, and J. Palmer, "Information survivability at Rome laboratory," *1997 IEEE Information Survivability Workshop*, 1997.
- [3] M. Namjoo, "Techniques for concurrent testing of VLSI processor operation," *Proc. International Test Conference*, pp. 461–468, November 1982.
- [4] P. Ammann, S. Jajodia, C. McCollum, and B. Blaustein, "Surviving information warfare attacks on databases," *1997 IEEE Symp. on Security & Privacy*, pp. 164–174, May 1997.
- [5] D. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 222–232, February 1987.
- [6] A. Clyde, "Insider threat identification systems," *Proc. 10th National Computer Security Conf.*, Sept. 1987.
- [7] 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.
- [8] 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.
- [9] J. Hochberg, K. Jackson, C. Stallings, J. McClary, D. DuBois, and J. Ford, "NADIR: An automated system for detecting network intrusions and misuse," *Computers and Security*, vol. 12, pp. 253–248, May 1993.
- [10] 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.
- [11] P. Porras and P. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," *National Information Systems Security Conf.*, pp. 353–365, Oct. 1997.
- [12] 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.
- [13] M. Schuette and J. Shen, "Processor control flow monitoring using signed instruction streams," *IEEE Transactions on Computers*, vol. C-36, pp. 264–276, March 1987.
- [14] S. Upadhyaya and B. Ramamurthy, "Concurrent process monitoring with no reference signatures," *IEEE Transactions on Computers*, vol. 43, pp. 475–480, April 1994.
- [15] D. Denning and P. Neumann, "Requirements and model for IDES - a real-time intrusion-detection expert system," *Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA*, August 1985.
- [16] D. Pradhan, *Fault tolerant computer system design*. Prentice-Hall, 1996.