An Analytical Framework for Reasoning About Intrusions[1]

Shambhu Upadhyaya and Ramkumar Chinchani
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, New York 14260
{shambhu, rc27}@cse.buffalo.edu

Kevin Kwiat
Air Force Research Laboratory
525 Brooks Road
Rome, New York 13441-4505
kwiatk@rl.af.mil

Abstract

Local and wide area network information assurance analysts need current and precise knowledge about their systems activities in order to address the challenges of critical infrastructure protection. In particular, the analyst needs to know in real-time that an intrusion has occurred so that an active response and recovery thread can be created rapidly. Existing intrusion detection solutions are basically after-the-fact, thereby offering very little in terms of damage confinement and restoration of service. Quick recovery is only possible if the assessment scheme has low latency and it occurs in real-time. The objective of this paper is to develop a reasoning framework to aid in the real-time detection and assessment task that is based on a novel idea of encapsulation of owner's intent. The theoretical framework developed here will help resolve dubious circumstances that may arise while inferring the premises of operations (encapsulated from owner's intent) by way of examining the observed conclusions resulting from the actual operations of the owner. This reasoning is significant in view of the fact that intrusion signaling is not a binary decision unlike error detection in traditional fault tolerance. Our reasoning framework has been developed by leveraging the concepts of cost analysis and pricing under uncertainty found in economics and finance. Our main result is the modeling of user activity on a computing system as a martingale and the subsequent quantification of the cost of performing a job to enable decision making.

1. Introduction

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. They are known to be largely firsthand and heuristic. In order to contain the damage effectively it is essential to detect intrusions concurrently so that recovery and restoration of service can be expedited.

We have developed a new host-based concurrent intrusion detection scheme (CIDS) [2]. Our scheme is based on the hypothesis that if one could reasonably encapsulate the intent of the user of a computing system, then it is possible to assess intrusions by monitoring the activities on-line. One way of realizing this idea is by means of verification of satisfiability of *computing premises* assisted by on-line monitoring of *conclusions* resulting from actual user actions. In addition, sequences of events and actions can be derived by analyzing the dynamically generated user data in real-time so as to enhance the intrusion assessment capabilities and lower the detection latency. Quick recovery is only possible if the assessment scheme has low latency and it occurs in real-time. Precise damage assessment can then be done by considering transaction and data dependency relationships.

In our technique of intrusion detection using verifiable assertions, flagging an intrusion is not a binary decision. Often, when reasoning backwards to the premises of operations from the set of observed conclusions**,** dubious circumstances (e.g., acceptance or rejection of a case) arise. While the idea of encapsulation of user's intent and the monitoring of user operation and profiling patterns gives a mechanism for intrusion detection, assessment and forecasting, the question of when to declare an anomaly as an intrusion needs further investigation. Any decisions on acceptance or rejection of a hypothesis and admittance or termination of service must be made rationally and on the basis of some reasonableness criteria; otherwise there is a

[1] Research supported in part by U.S. Air Force Research Laboratory, Rome, New York, under Contract: F30602-00-10507

danger of excessive false alarms. The questions that need to be addressed are: What is a reasonable plan when a user initially provides his session-scope to the security monitor? When is it reasonable to suspect a user as an intruder? Can reasonableness checks be used to flag non-deliberate faults such as operator faults in a given system?

We leverage concepts from other domains such as artificial intelligence, business and finance to build a reasonableness check framework. In business and marketing domains, there are many instances where decisions have to be made, even though the exact details of an operation or that of an event happening is not known beforehand. In such a case, decisions have to be made on the basis of the cost involved and the probabilities of those events happening. Although the course of events is unknown, the possible events that could occur are enumerated and its probabilities and costs evaluated. This process is referred to as Risk Analysis [3]. Analytical models exist for reasoning on pricing under uncertainty [4]. Some of these principles will be employed in our reasoning framework to provide an analytical basis for making rational decisions.

In Section 2, our basic intrusion detection system is briefly reviewed to facilitate the presentation of our analytical model of reasoning about intrusions. The necessary abstraction and formalism of system components appear in Section 3. The cost analysis basics are given in Section 4 and a stochastic modeling of user job activity on a computer system is developed in Section 5. The reasoning framework and the associated algorithms are discussed in Section 6, preliminary results are given in Section 7 and conclusions appear in Section 8.

2. Review of CIDS

The framework presented in this paper is aimed at detecting and apprehending intrusive behaviors such as masquerading, legitimate user penetrations, internal abuse, illegal resource access etc. It cannot directly detect denial of service attacks unless they occur through a well-defined user session [2]. Techniques to deny generic DOS attacks are discussed elsewhere and since DOS attacks are not treated as intrusions, they are not discussed here any further.

Concurrent intrusion detection typically proceeds at the user operation level. However, there is a need for a reference graph to compare the user operations with at this level of abstraction. Therefore, we devised the method of encapsulation of user's intent [2] by querying the user for a session-scope [2]. The user gives a summary of his intended system usage. Once the scope file is submitted, the user is allowed to continue with his session. This session scope remains active until the user proceeds to alter it after proper authentication and hence making it less onerous for the user. Meanwhile the system translates the scope file into a sprint (Signature Powered Revised Instruction Table) plan. When no strict ordering of events is possible on the activities of the user, the sprint plan is simply a table of *verifiable assertions*. There is no control flow information as such. A verifiable assertion is formally defined as a quadruple, which associates a user with the intended operation over a given period of time. Its format is *(subject, action, object, period),* where *subject* is a user, *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 an action.

The assertions give a mechanism for monitoring the user behavior. Verifiable assertions are a generalization of IDES's [6] 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. There is only one monitor process per user per system even when multiple sessions are opened.

When the user is in session, the monitor process monitors the user commands and checks if the command is the one he originally intended to execute. Certain tolerance limits are set to take care of unplanned but benign deviations. Any significant deviation from the plan is an indication of potential intrusion. Fig. 1 gives a self-explanatory flow diagram of the basic concurrent intrusion detection system (CIDS).

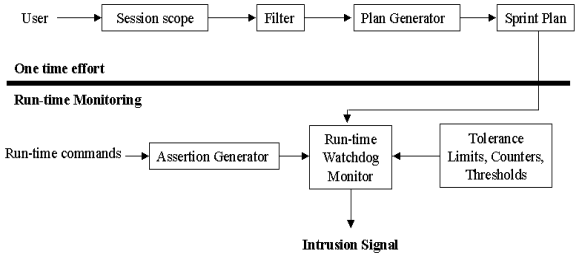

Fig. 1. Flow diagram of CIDS

CIDS can be implemented in a hierarchical way like several other intrusion detection efforts [7], [8]. An algorithm and several intrusion scenarios and their detection have been reported in [2]. A prototype of the basic CIDS has also been implemented and several simulation experiments conducted [9].

The basic CIDS described here has limited detection capability since only a set inclusion check is performed on the assertions in the sprint plan. Intrusions can easily be forced by concocting malicious sequences of operations, despite staying within the scope of operations. Whereas CIDS provides a novel way of intrusion detection, its detection

[2] Encapsulation of user intent should not be confused with user intent modeling [5]. Unlike inferring user's intent, our technique actively queries the user for his intent. This data is generally small and makes the intrusion detection problem tractable.

limitation can be addressed by discovering patterns (e.g., certain types of combinations of assertion metrics) in the user sprint plan, user generated data and network data records. Pattern profiles can be leveraged to enhance the anomaly detection, intrusion identification, recovery, and intrusion forecasting potential of our technique. In the rest of this paper, we present the sketch of a formal reasoning framework to address intrusion detection and assessment based on user operation monitoring supported by pattern profiling.

3. Abstractions and Formalisms of System Components

In order to develop a reasoning framework for intrusion detection, it is necessary to first develop formal definitions for key system components such as a User, Resource, Action, Operation, Object, Event, Sequence and Job with the appropriate security and abstraction levels. A complex system can usually be broken down into a set of modules, and each of these modules can further be broken down into simpler modules and so on. Each such level represents a level of abstraction.

3.1. Definitions of Basic Entities

User: A *user U* is defined as one who gains access to the system resources by a userID and password submission. A user could be an authorized subject or an intruder. A user is denoted by a 2-tuple $U = (I_U, L_U)$, where I_U is a subjectID unique to the user and L_U is the security level of the user. L_U could also be considered to be the group to which the user belongs.

Resource: A *resource R* is defined as a system entity whose services are used when a job is being performed. A resource could be complex consisting of various levels of abstraction. It is denoted by a 2-tuple $R = (I_R , Al_R)$, where I_R is an identifier unique to the resource *R* and Al_R is a set of 2-tuples (A, l), where *A* is the set of all valid actions at the level of abstraction *l*.

Action: An *action A* is defined as a basic operation. Actions are classified as accessors and modifiers. Accessor actions do not change the state of the resource, but modifier actions do. An example of an accessor action is *ReadByte* and an example of a modifier action is *WriteByte* when actions performed are on bytes.

Operation: An *operation O* is a generalization of *action* and is defined as an action that is performed on an object belonging to a resource at a particular level of abstraction. An operation may be broken down into two or more sub-operations, which are operations by themselves. An operation is denoted by a 2-tuple $O = (A, K)$, where *A* is the action performed on some object *K*.

For the purpose of intrusion detection, the floor level of operation is defined at the user command level.

Object: An *object K* is defined as an entity belonging to some resource at a specified level of abstraction and on which an operation is performed to get the job done. It is denoted as a 3-tuple $K = (I_K, R_K, l_K)$, where I_K is the objectID, R_K is the resource to which it belongs and l_K is the level of abstraction at which it is defined. Typically, operations are performed on a file. A file can be represented as *(ThisFile, ThisFileSystem, File)*.

Event: An *event E* is an operation associated with time and is defined as an operation performed at some specific time instant. It is denoted by a 2-tuple $E = (O, t)$, where *O* is an operation performed and *t* is the time at which this operation is performed.

Sequence: A *sequence S* is a set of events, denoted by $S = (E_1, E_2, E_3, ...)$, where E_i are events forming the states of the sequence. It is possible that events could repeat over time. In other words, there could exist events E_i, E_j expressed as tuples (O_i, t_i), (O_j, t_j), such that $O_i = O_j$ for $t_i \leq t_j$.

Job: A *job J* is defined as a service that a user expects the system to perform. A job by itself is complex and can be broken into sub-jobs or operations. There could exist multiple levels of abstraction. A job *J* is denoted by a 3-tuple $J = (O, S, U)$, where *O* is the set of operations performed, *S* is the sequence of operations, and *U* is the user on whose behalf the job is being performed.

We now define the levels of abstraction in resources and jobs.

3.2. Levels of Abstraction

Resource Abstraction: Consider the disk storage. It can be divided into individual disks, each disk into partitions, each partition into a file system, and each file system into its constituent structures like files, directories, inodes, and eventually each of these into bytes. Actions can be defined at each of these levels. At the byte level, the set of actions could be (*ReadByte, WriteByte*). At the file level, the set of actions could be (*ReadFile, WriteFile, CreateFile, DeleteFile, TruncateFile, …*).

Job Abstraction: A user gains access to a system in order to get some jobs done. The outcome of getting a job done may be the normal completion or a system compromise. Let us consider a simple job, a LaTeX job. It can be broken down into an *editing* job and some LaTeX commands. The constituent operation *O* in the editing job can be specified as $O = (EditFile, ThisFileObject)$, which means, the action is *EditFile*, on object *ThisFileObject*. For greater granularity, *O* can be broken down into smaller operations involving objects at lower levels of abstraction for that resource, i.e., *ThisFileObject* at a lower level would be equivalent to some file structures and data bytes and *EditFile* would now be broken down appropriately as *ReadByte* or *WriteByte*.

3.3. Validity of an Operation

An operation $O(A, K)$ is invalid if:

1) Action *A* is not defined on objects of type *K*.

2) The user is not permitted by the system policy to perform the action *A* on object *K*.

3) The object *K* does not exist.

The user is penalized for performing an invalid operation and the penalty depends on the operation and how it affects the system.

3.4. Expected Sequences of Operations

Whenever a system is built, it is built with a goal and a purpose in mind. In order to make use of the system, it must be used in proper ways. This forms the basis for expected operations in an expected sequence. There need not be a unique sequence to complete a job, but the sequences can be rated from the most normal to the least normal way of getting the job done. So, there are usually a few sequences, which constitute normal job behavior, and the rest are deviations from this behavior. Since the user intent is encapsulated at the beginning of a session, it is possible to deduce the most normal sequences of operation. These sequences develop a default profile of the given job. We now delineate a method to formulate the expected sequences of operations.

Let the set of all possible sequences of operations be $\boldsymbol{S}$. Assume that it consists of the expected sequences $S_1, S_2, \dots, S_m$ and a pseudo sequence S', which represents all the unknown sequences. S' abstracts the anomalies and can be assumed to consist of sequences longer than the longest of $S_1, S_2, \dots, S_m$ since any anomalous sequences are substantial deviations of expected sequences.

$$\boldsymbol{S} = \{S_1, S_2, \dots, S_m, S'\} \quad (1)$$

$$P\{S'\} = 1 - [\, P\{S_1\} + P\{S_2\} + \dots P\{S_m\}] \quad (2)$$

Similarly, let the set of all possible operations be $\boldsymbol{O}$. Let the elements of the set be $o_1, o_2, \dots, o_k$ and a pseudo operation o'. This pseudo operation o', like S', represents all the unknown operations which are not present in the sequences $S_1, S_2, \dots, S_m$. o' can be relaxed by inferring more operations which could be anomalies or mistakes. Such operations would then no longer be counted as unknown.

$$\boldsymbol{O} = \{o_1, o_2, \dots, o_k, o'\} \quad (3)$$

$$P\{o'\} = 1 - [\, P\{o_1\} + P\{o_2\} + \dots + P\{o_k\}] \quad (4)$$

We now define a sequence *S*, which becomes the profile of a particular job of a particular user. Let O_i be the operation performed at state *i* of the sequence *S*. The probability distribution of operations at each state of this sequence *S* can be found using the above information and the theorem of Total Probability. According to our scheme, all anomalies need not be individually modeled. Only the knowledge of the expected sequences and the associated operations is needed and these sequences and operations are usually finite. The monitoring of operations is done at the user command level, and these sequences are sequences of commands a user issues. Each command, however, may be monitored at a greater granularity if the command allows for excessive freedom and control to the user.

It may be noted that these sequences do not represent the attack signatures as in other intrusion detection systems. Also, there could be a finite but large number of partial orderings in the set of operations, which may cause potential state explosions. This may lead one to think that rapid intrusion detection is impractical and the large number of possibilities may result in non-determinism and false positives. The idea of obtaining a session scope from the user at the beginning of the session and prioritized workspace allocations will alleviate this problem. This is part of our ongoing work, and the details are outside the scope of this paper.

4. Cost Analysis

4.1. Assigning Costs to Operations

Performing some operations on appropriate resources in some order or sequence can complete a job. When an operation is performed, a cost is calculated based on the use of resources. The cost can be calculated at each level of abstraction of the resource for greater granularity.

The C*ost of an Operation, C_o* is defined as a function of the amount of the resources being used to perform the operation. It is necessary to monitor the usage of resources when the operation is being performed because even though there may be slight deviations from the intended operation, the user could still dominate the resources, reducing the quality of service (QoS) for other users. This may not qualify as an intrusion, but it is still an abuse of resources and hence an anomaly. The method of calculating this cost of performing an operation is discussed in Section 6.2.

4.2. Assigning Costs to Sequences

The sequence of operation performed to complete a job on a given system is not unique. Depending on the user's intent [2], a set of expected sequences and the associated expected operations are developed as described in Section 3.4. When the user logs into the system and submits jobs, the behavior is matched with the one that is predicted. A measure of the deviation from the expected behavior is called the *Cost of Deviation, C_d.*

A genuine user would conform to his user profile and will have very little deviation. On the other hand, an intruder who may break into a user's session is normally not aware of the user's profile and is expected to deviate from this profile. The method of calculating this cost is discussed in Section 6.3.

4.3. Computing the Cost of Performing a Job

The Cost of performing a job is calculated as the cumulative cost of operations involved at each instant of time

in a sequence along with the penalty for deviations from the expected sequence.

$$Cost(J) = \sum [C_o(O_i) + C_d(O_i),] \quad (5)$$

summed over a sequence S_j followed by the user.

5. Stochastic Modeling of Job Activity

The choice of an operation to perform a certain job is probabilistic and hence is a random variable X(ξ), where ξ is the operation chosen. The set of operations is finite. A sequence of any such operations has a temporal property making it a discrete-stochastic process denoted by X(t, ξ). The operations performed over time t form a sequence of random variables $X(t_1, \xi)$, $X(t_2, \xi)$, … $X(t_m, \xi)$. Since it is more relevant for a profiler to know which operation follows which other and not the instant of time itself at which an operation is performed, the parameter t_i here is made to denote a state in the sequence rather that time; t_1 is the first state in the sequence, t_2 the second, and so on. In summary, this stochastic process represents the sequence of operations performed while completing a job.

A user can perform a specific type of job zero or more times in one user session. A stochastic process is associated with a job every time it is executed. But for the purpose of assessment and prediction of operations (intrusions) a user can perform, a history of processes associated with the type of job is used. For a specific job, we define a sequence of stochastic processes, $X_1(t, \xi)$, $X_2(t, \xi)$, …$X_n(t, \xi)$ to represent the history of sessions related to that job.

The lateral sequence of random variables, $X_1(t_1, \xi)$, $X_2(t_1, \xi)$, … $X_n(t_1, \xi)$ forms the user job profile history at state 1 of the sequence of operations. In general, a lateral sequence of random variables $X_1(t_i, \xi)$, $X_2(t_i, \xi)$, … $X_n(t_i, \xi)$ forms the user job profile history for state i of the sequence of operations the user chooses to complete the job.

5.1. The Concept of a Martingale [10, 11]

Let X_1, X_2, … X_n be a sequence of random variables defined on a common probability space $(\Omega, \mathfrak{I}, P)$ and let $\mathfrak{I}_1$, $\mathfrak{I}_2$, … $\mathfrak{I}_n$ be a sequence of σ-fields all belonging to $\mathfrak{I}$. The sequence $\{(X_n, \mathfrak{I}_n), n=1, 2, \ldots\}$ is a *martingale* if for each *n* it satisfies the four conditions below:

(1) $\mathfrak{I}_n \subset \mathfrak{I}_{n+1}$; (6)

(2) X_n is measurable $\mathfrak{I}_n$; (7)

(3) $E(|X_n|) < \infty$; and (8)

(4) $E[X_{n+1} | \mathfrak{I}_n] = X_n$ with probability 1. (9)

The martingale theory uses concepts of conditional probability and has found a lot of applications in the economics and finance domains [Malliaris 91]. We elaborate this further since we are going to model user activity as a martingale. This model will help us to draw a parallel between the uncertainties in intrusion detection and the concepts of pricing under uncertainty in economics.

Pricing under Uncertainty or *Futures Pricing*, under certain conditions is a martingale [Malliaris 91]. Let X_1, $X_2 \ldots X_t$ be a sequence representing the time sequence of prices till time t. We assume that these random variables are bounded because the prices of a commodity are finite and usually have some upper bound. We also assume that an economic agent is observing the prices of these commodities, and this agent is predicting the futures price based on the past information contained in the σ-fields generated by these random variables. The futures price of a commodity depends only on the last known distribution and not the entire history of the prices. This result is proved in [Malliaris 91]. The martingale property has also been used to predict other market parameters like a share of a stock, risk evaluation, etc.

5.2. User Activity as a Martingale

Theorem: Let the lateral sequence of random variables for any state *i* of a sequence of operations be denoted as $X_1(t_i, \xi)$, $X_2(t_i, \xi)$, … $X_n(t_i, \xi)$. Such a sequence of user activity is a martingale.

Proof: Since the parameters t_i, ξ, remain constant, we simplify the notation for the sequence as $X_1, X_2, \ldots, X_n$. Our goal is to predict the next distribution, given this history.

Let us assume that $X_{n+1} = X_n + A_n$, where A_n is an independent identically distributed random variable with zero mean and a small variance. This assumption is justified by the fact that we expect the user to perform operations not very different from the last time. Any small deviation that could possibly occur is embodied in A_n.

$\mathfrak{I}_n = \sigma(X_n, X_{n-1}, \ldots X_1) \subset \mathfrak{I}_{n+1} = \sigma(X_{n+1}, X_n, \ldots X_1)$, since the information embodied in the sequence $X_{n+1}, X_n, \ldots X_1$ is larger than that in $X_n, X_{n-1}, \ldots, X_1$. This satisfies condition 1 of martingale (see Eq. (6)).

Since the set of operations is finite and we define an initial distribution based on the encapsulation of users intent and other general information regarding user behavior, which involves every outcome in $\mathfrak{I}$, it follows that X_n is measurable $\mathfrak{I}_n$, satisfying condition 2.

Also, every outcome is assigned a finite value. This satisfies condition 3.

We establish condition 4 as follows.

$E\{X_{n+1} | X_n, X_{n-1}, \ldots X_1\}$

$= E\{X_n + A_n | X_n, X_{n-1}, \ldots X_1\}$

$= E\{X_n | X_n, X_{n-1}, \ldots X_1\} + E\{A_n | X_n, X_{n-1}, \ldots X_1\}$

$= X_n + E\{A_n\}$ (since A_n is an independent random variable)

$= X_n + 0$ (since $E\{A_n\} = 0$)

$= X_n$ (10)

Development of this theorem provides a breakthrough for our reasoning since we are able to integrate the user activity profile in a quantitative framework. This result allows us for

the quantification of measures leading up to intrusions and its evaluation as described in the next section.

6. The Reasoning Framework

We define two thresholds T_l and T_h on the cost of performing a job in order to facilitate the reasoning process. As the job is being performed, the cost keeps accumulating in a monotone non-decreasing manner. Three regions of costs, viz., non-intrusive, indeterminate and intrusive are defined using the two thresholds T_l and T_h as shown in Fig. 2.

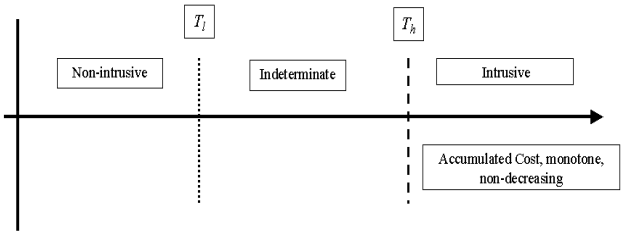

Fig. 2. Illustration of thresholds and cost buildup

The cost of performing a job maps into these regions. The mapping is done actively, following the sequences as the events occur.

$$Cost\ (non\text{-}intrusive) \leq T_l \leq Cost\ (indeterminate) \leq T_h \leq Cost(intrusive) \quad (11)$$

The above relation alludes to the relationship between the costs and regions demarcated by the thresholds.

6.1. Reasoning Basis

This section discusses the basic method used for developing the reasonableness test framework. The quantification of parameters involved is discussed in the following sub-sections.

As a user begins to perform operations to complete a job, he begins to incur costs of both types as discussed in Section 4.3. Clearly, when the accumulated costs map into the non-intrusive region, an intrusion will not be signaled. On the other hand, when the costs map into the intrusive region, an intrusion will be signaled. When the costs map into the indeterminate region (window of uncertainty) the situation is ambiguous and needs to be resolved. We have the indeterminate region since it is not possible to make a binary decision whether a user's activity is intrusive or not. When the costs map into the indeterminate region, the goal would be to reduce this window of uncertainty to the greatest possible extent, making the decision-making process more deterministic and quicker. This would arrest any further damage if the user were really an intruder.

An initial default profile is created when the user submits his intent in his first login session [2]. Subsequent job and user sessions shape this initial profile. Based on this profile, we predict what the user's behavior would be and compare the actual behavior to this predicted behavior. When an intruder breaks into someone's account, he would hardly know the user profile of the person's account he has broken into. His actions will appear as a deviation from the predicted user profile. The accumulated costs in such cases are likely to map either into the indeterminate region or the intrusive region. Genuine users do make mistakes at times, but they correct themselves and conform to the profile. Therefore, it can be expected that the accumulated costs would map into the non-intrusive or indeterminate regions. Once in the indeterminate region, we use factors such as cost gradients to shrink this region. This process may have to be repeated until this indeterminate region is very small and there are effectively only two regions, viz., non-intrusive and intrusive. When that is done, any further mappings would be resolved as either non-intrusive or intrusive.

6.2. Quantifying the Cost of Operations

The number of resources in a system is finite and the number of actions that can be performed on a resource is also finite. Actions at higher levels are defined in terms of actions at lower levels (see Section 3). So we start at the lowest level of abstraction and assign costs for these basic actions. The cost of each action at a particular level may depend in turn on the semantics of the operation, i.e., whether the action is an accessor or a modifier. We then progress upward defining costs at higher levels based on the ones defined at the lower levels. For example, take the resource *Hard Disk*. The lowest logical level is that of a *Byte* or some similar data type. The actions that are defined at that level could be *ReadByte* and *WriteByte*. Since the semantics of *ReadByte* would be just reading a *Byte* without any loss or damage to data, it has a smaller cost. *WriteByte* on the other hand, though it operates on a *Byte*, has a higher cost because it could result in damage or loss of data. At the next higher level of a *File*, we could have the actions *CreateFile*, *ReadFile*, *WriteFile*, *DeleteFile*. Each of these actions in turn uses the lower level actions *ReadByte* and *WriteByte*. Moreover, reading five *Bytes* would have more cost than reading one *Byte*, because it is equivalent of performing the *ReadByte* action five times. It is also possible that the operation cannot be completed because it is an invalid one (ref. Section 3.3). This can also be interpreted as the operation can be completed in infinite time. Such operations are assigned a very high cost. This is additionally justified by the fact that choice of an invalid operation indicates deviation. The goal of this scheme is to reflect the amount of resources being used.

The cost of performing an operation reflects the usage of resources. An abuse of the resources causes accumulation of high costs, eventually leading to abuse detection. It should be noted though that intruders are subtler and rarely do they

resort to behaviors indicative of resource abuse. Therefore, this component remains negligible in such cases.

6.3. Quantifying the Cost of Deviation

The encapsulated user's intent and the actual choice of operations in each session form a list or history of profiles. This history is used to model the future user behavior. A non-linear mean square estimation method [12] can be used for this purpose.

Let the sequence of random variables X_n, X_{n-1}, X_{n-2}, … , X_1 represent a history of choice of operations at some state i of sequences followed in the respective sessions.

We make the futuristic estimate as:

$$E\{X_{n+1} \mid X_n, X_{n-1}, \ldots X_1\} = X_n \quad (12)$$

Now, since we have modeled the future user behavior, we define the following relation to quantify the deviation from the normal user behavior.

$$C_d(O_i) \propto [E\{X_n\} - X_n(O_i)] \quad (13)$$

where O_i is the operation chosen.

$$\text{or } C_d(O_i) = \lambda \,.\, [\, E\{X_n\} - X_n(O_i)] \quad (14)$$

where λ is the constant of proportionality. It could also be used as a scaling factor to map the cost into the regions defined in Fig. 2.

The following algorithm is used to define the initial distribution for the operations associated with completing a job. This algorithm, based on the development in Section 3.4, takes into consideration the sensitivity of the cost of deviation to the choice of operations. This initial default profile is based on the user intent provided at the start of the very first user session.

Algorithm: INIT_DISTR

1) Let **S** = $\{S_1, S_2, \ldots, S_m\}$ be the set of sequences (ref. Section 3.4).
2) Sort **S** in decreasing order of the lengths of the sequences.
3) Let S_{max} be the longest sequence in **S**.
4) Associate a stochastic process X(t, ξ) with S_{max}.
5) At each state t_i of this stochastic process, enumerate the operations and assign probabilities to each of these operations (see Section 3).
6) Let R be a large real number. /* The actual mapping of each random variable $X(t_i, \xi)$ begins here */
7) for i = 1 to $|S_{max}|$ do /* where |x| means the length of x */

7.1) Let **O** = $\{o_1, o_2, \ldots, o_k\}$ be the set of operations at state i, sorted in the decreasing order of the probabilities.

7.2) $X(t_i, o_1) = R$

7.3) for j = 2 to size(**O**) do

7.3.1) $X(t_i, o_j) = R + (-1)^j [\mathbf{f}(P\{o_1\} - P\{o_j\})]$

7.3.2) j = j + 1

A pictorial representation of the initial distribution resulting from the above algorithm at each state is given in Figure 3.

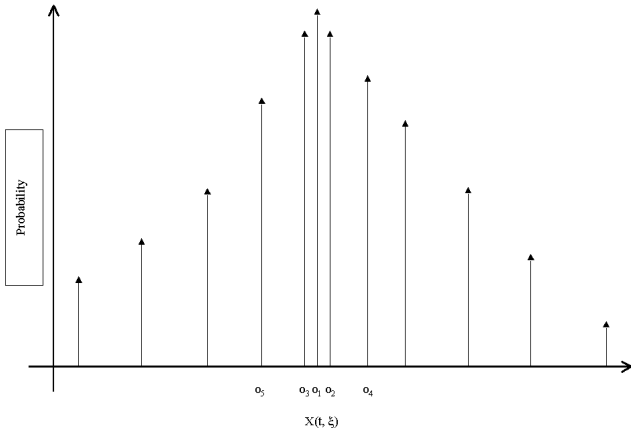

Fig. 3. The initial distribution

The distribution is shaped by the function **f** in the algorithm in such a way that the operations with higher probability of selection by the user lie closer to the mean and others are mapped away from the mean. This would allow the distribution to be more sensitive to deviations (see (14)). The operation with the highest probability is used as a reference. The mapping of other operations is defined by the function of the difference of their probabilities from this reference probability. This function **f** could be a linear or exponential function and is chosen appropriately.

Since, at any step of the algorithm, only a finite and small number of values are being handled irrespective of the user input, the time complexity of this algorithm is O(1).

The above procedure determines the initial default user profile for a particular job. This profile is subject to change based on the user's behavior, i.e., the distribution at each state of this sequence is influenced by future choices of operations. We now propose an algorithm, which modifies and updates the user profile X(t, ξ) to accommodate the current user activity.

Algorithm: MODIFY_DISTR

1) Let X(t, ξ) be the stochastic process representing the user profile.
2) Let S be the sequence the user follows to complete a job.
3) if $Cost(J) > T_h$, exit, where *J* is the job performed till its completion or till flagging of intrusion.
4) else if |(X(t, ξ)| < |S|, extend X(t, ξ) by |S| – |X(t, ξ)| states and associate random variables for each of these states using INIT_DISTR
5) for i = 1 to |(X(t, ξ)| do

5.1) update the probability distribution for $X(t_i, \xi)$, considering that unknown operations are represented by o' (ref. Sec. 3.4).

This algorithm is simple and all it does is update the frequency distributions at each state of the stochastic process

X(t, ξ). The complexity of the operations depends linearly on the length of the sequence of operations as chosen by the user. Hence, the complexity equals O(n).

6.4. Quantifying the Thresholds T_l anf T_k

Setting the thresholds to determine the intrusive and non-intrusive regions is not easy. Therefore, we use certain bounds to initially set the thresholds. These thresholds will be refined subsequently based on the user operations.

Clearly, a job can be completed in multiple ways, with each having its own sequence of operations. Let S_{min} represent the shortest of such sequences and S_{max} the longest one. Threshold T_l is the lowest amount of costs accumulated over the sequence S_{max}.

$T_l = Cost(J) = \Sigma [\alpha . C_o(O_i) + \beta . C_d(O_i)]$, summed over the sequence S_{max}. (15)

Here, $C_o(O_i)$ and $C_d(O_i)$ are the minimum costs, or in other words, an appropriate operation O_i is chosen such that these costs are the minimum. Any other sequence will accumulate smaller costs assuming the user is conforming to his profile. Therefore, the costs accumulated in such a case will remain in the non-intrusive region. A reasonable amount of error ε can be added to relax the bound.

The threshold T_h is the maximum costs accumulated over the sequence S_{min}.

$T_h = Cost(J) = \Sigma [\alpha . C_o(O_i) + \beta . C_d(O_i)]$, summed over the sequence S_{min}. (16)

Any other sequence (ref. *S'* in Section 3.4) will accumulate higher costs, indicating that the user has been deviating from his profile in an intrusive way. Hence, costs accumulated in such a case will overflow into the intrusive region. Just as in the case of T_l, a reasonable amount of error can be added or subtracted from this bound.

6.5. Decision-making in the Indeterminate Region

If the cost of performing a job maps into the indeterminate region at any time, the situation needs to be resolved quickly to ensure concurrent detection. We propose to handle this by dynamically moving the thresholds based on certain criteria. The cost may have mapped to the window of uncertainty due to genuine, but inadvertent deviations by the user. With this optimistic view, the T_l threshold is moved toward T_h. The amount by which it is moved is equal to the minimum costs that would be accumulated from this current state till the completion of the job over the longest sequence possible from this current state. This is a recursive procedure (see Section 6.4).

T_h needs to be moved towards T_l since there is an appreciable deviation from the profile. The idea is to shrink the window of uncertainty so that a speedy decision can be taken. This movement depends on the rate at which the costs have been accumulated since the last time the threshold movement was triggered. A genuine user does occasionally make mistakes, but would otherwise conform to his profile and hence, the gradient of costs is expected to be small. An intruder on the other hand would perform largely deviating operations in a short span of time, causing a greater gradient of costs. In such a case, the window of uncertainty should be narrowed sharply.

$(T_h - T_l) \propto 1 / G$ (17)

$(T_h - T_l) = \Upsilon \times [1 / G]$, where Υ is a constant (18)

where $G = (\Delta Cost)/n$, where $\Delta Cost$ is the difference in costs accumulated since the last time the threshold movement was triggered, and n is the number of states traversed since that time.

The threshold movement, which is the core of our reasoning, is presented in the form of an algorithm. Whenever the costs map into the indeterminate region, this algorithm is invoked.

Algorithm: DECIDE

1) Let S be the longest sequence of operations that is needed to complete the job from the current state in the user's sequence of operations.
2) Calculate c = minimum cost over S.
3) $T_l = T_l + c$
4) Let C = currently accumulated costs
5) Let C' = accumulated costs when DECIDE was last invoked. If DECIDE was never invoked C' = 0.
6) Calculate gradient of costs G = [C – C'] / n, where n is the number of states traversed since DECIDE was last invoked.
7) $T_h = T_l + \Upsilon \times [1 / G]$, where Υ is a constant.

This algorithm is used only when the accumulated costs map into the indeterminate region. In the best case, this algorithm may not be invoked even once. In the worst case, the algorithm may have to be invoked for the entire length of the sequence. The time complexity would then be O(n).

7. Preliminary Results

The above framework was tested by integrating it into the basic CIDS and a summary of these results is presented. These tests are by no means exhaustive and serve merely as a proof of concept. Moreover, there is a lack of proper test data at our level of monitoring, requiring us to generate some of our own. In reality, it calls for more specialized data specific to an environment this system would be deployed.

We have simulated a "Virtual Banking Environment" for testing purposes. This environment consists of entities such as *Bank*, *Account*, *Employee*, *Customer*, etc. There are multiple employees in a bank at various levels in the hierarchy and each with a specific job detail. Customers can have multiple accounts in the bank and each employee typically has access to multiple accounts. Considering the

value of the information, there is always a possibility of security policy violations or attempts thereof.

Any user connecting to the bank for the first time is presented with a GUI, which consists of a set of jobs based on his userID. The user can now choose a subset of these jobs and this forms his intent. The system remembers his previous intent, if any, and the user can choose to agree with the old one.

User activity is primarily in the form of GUI events, and each event is bound to a specific action or command. These events in time domain represent a sequence of operations.

Monitoring is primarily performed by a master watchdog that continuously monitors user activity in the system. Accesses to the database are monitored by a file watchdog, which is built around the system database. It communicates with the master watchdog to report any violations or events.

The system was designed to be as realistic as possible and was built on CORBA [14] to support the distributed architecture of the banking environment. The system was primarily developed in Java. Information is stored in an Oracle database and JDBC provides connectivity to this database. It uses SQL to perform queries on this database. All GUI components are written to be lightweight and small using JFC Swing [15].

Sequences which are perfectly valid and those resulting in a possible compromise are noted and fed to the monitor. These form the input to the watchdogs.

We used system overhead and detection coverage as performance metrics. There is a fixed amount of processing and memory overhead due to the initial setup of the watchdog processes, and the increase is an approximate linear function of the number of users currently accessing the system. This is due to the fact that additional processing is attributed to the threads, which the master watchdog launches to monitor per user activity. There is very little permanent storage required because of the martingale approach. This system gives a very good coverage with few false positives when the user doesn't resort to excessively deviant behavior. This could be a part of the policy enforcement. Any significant exploratory behavior is identified by the system as "noise" and this may result in large false positives.

8. Discussion

We have presented a basic reasonableness test framework for detecting intrusions before they manifest in harmful ways and cause irreparable damage. The low latency of detection is a significant feature of our technique. The underlying idea of intrusion detection is based on a novel concept of encapsulating owner's intent. Obtaining the owner's intent prior to starting a session makes the concurrent intrusion detection tractable unlike the approaches that work with huge audit data. This paper is mainly focusing on the development of an analytical framework for the reasonableness test part of intrusion assessment. Most of the reasonableness test works in the literature are ad hoc [13] in nature. The framework presented here gives a systematic way of checking if a particular decision is made in a rational way. This framework is applicable not only for reasoning about intrusions but also to situations where decision-making is not a binary process. The major contribution of our work is the integration of the user activity profile into a quantitative framework. We have leveraged concepts from economics and finance to accomplish this goal.

Most existing security monitoring systems are rule-based, which means that they cannot detect unseen intrusions. Unlike checking for an ever-growing set of intrusive sequences, our approach monitors to see that the user follows a plan. Moreover, since we work with the concept of jobs, and solicit a session scope, we have smaller and focused set of operations to deal with. Maintaining two thresholds enables us to make a more informed decision with reduced false positives. We are currently working on devising techniques to manage and reduce the state explosion and false alarms while handling various sequences of operations. A preliminary prototype of the systems has been setup and studied [9] and the results obtained are encouraging.

Security has to be always linked with people. A user on a system implementing the proposed security monitoring framework must not resort to random and exploratory behavior. In other words, we assume that the users are well trained and will not behave like intruders or hackers. Since our system maintains history and profiles, it could be adapted to provide corrective measures too, in case of operator faults.

This system does have its limitations. Excessive exploratory behavior or "harmless" noise can result in large false alarm rates. Since the monitoring is done at a user command level, it cannot detect low-level network based attacks or external DOS attacks.

The reasoning framework and the concurrent intrusion detection ideas presented here are not replacements to existing core anomaly detection techniques. Instead, it can be used as a third-party module to supplement the current security systems. Based on the theoretical model developed here, we plan to implement a fully developed intrusion detection and assessment prototype in the future. This prototype will be subjected to tests in several environments. The factors influencing the parameters described in the paper will be studied by simulating intrusion scenarios and analyzing the results.

References

[1] T. Lunt, "A Survey of Intrusion Detection Techniques", *Computers and Security*, Elsevier Science Publishers Ltd., vol. 12, 1993, pp. 405-418.

[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*, July 1999, pp. 369-376.

[3] A. Mallik, Engineering Economy with Computer Applications, *Engineering Technology Inc*, 1979.

[4] S. Jagpal, Marketing Strategy and Uncertainty, *Oxford University Press*, 1999.

[5] T. Spyrou and J. Darzentas, "Intention Modelling: Approximating Computer User Intentions for Detection and

Prediction of Intrusions, *12th IFIP Information Security Conference*, 1996.

[6] 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**.**

[7] P. Neumann and P. Porras, "Experience with Emerald to Date", *1st Usenix Workshop on Intrusion Detection and Network Monitoring*, April 1999.

[8] J. Balasubramaniyam, J. Fernandez, D. Isacoff, E. Spafford and D. Zamboni, "An Architecture for Intrusion Detection Using Autonomous Agents", *Proceedings of the 4th IEEE Computer Security Applications Conference*, December 1998.

[9] K. Mantha, R. Chinchani, S. Upadhyaya and K. Kwiat, "Simulation of Intrusion Detection in Distributed Systems", *SCS Summer Simulation Conference*, July 2000.

[10] A. Malliaris and W.A. Brock, Stochastic Methods in Economics and Finance, *Elsevier Science Publishers*, New York, 1991.

[11] R. G. Gallager, Discrete stochastic processes, *Kluwer Academic Publishers*, 1995.

[12] A. Papoulis, Probability, Random Variables, and Stochastic Processes, Third Edition, *WCB/McGraw-Hill*, 1991.

[13] D. Pradhan, Fault-Tolerant Computer System Design, *Prentice Hall*, 1996.

[14] Jeremy Rosenberger, CORBA in 14 Days, *Sams Publishing*, 1998.

[15] S. Pantham, Pure JFC Swing, *SAMS*, 1999.