Using Attack Trees to Identify Malicious Attacks from Authorized Insiders

Indrajit Ray and Nayot Poolsapassit

Colorado State University,
Fort Collins, CO 80523, USA
{indrajit, nayot}@cs.colostate.edu

Abstract. A major concern for computer systems security is the threat from malicious insiders who execute perfectly legitimate operations to compromise system security. Unfortunately, most currently available intrusion detection systems (which include anomaly and misuse detection systems) fail to address this problem in a comprehensive manner. In this work we propose a framework that uses an attack tree to identify malicious activities from authorized insiders. We develop algorithms to generate minimal forms of attack tree customized for each user such that it can be used efficiently to monitor the user's activities. If the user's activities progress sufficiently up along the branches of the attack tree towards the goal of system compromise, we generate an alarm. Our system is not intended to replace existing intrusion detection and prevention technology, but rather is intended to complement current and future technology.

1 Introduction

Intrusion detection systems are an important tool to system administrators in their fight against malicious attacks on systems. These tools monitor different system activities and report on anything that can be construed as malicious. The system administrator looks at the reports generated by the intrusion detection system and (based on experience to some extent) determines which of the activities are malicious. In this work, we propose a quantitative approach to help system administrators make sound judgements regarding ensuing attacks. Our system is intended to complement existing tools to fight the war against crackers.

Existing intrusion detection systems suffer from two shortcomings. First, not many of them do a good job in handling threats from malicious insiders. These attacks, which are often considered to cause the majority of security breaches, can arise in one of two ways: (i) A user uses perfectly legitimate operations to exploit known system vulnerabilities and launches an attack. (ii) A user uses information and resources that do not fall directly under the category of computer system resources, and launches attack. An example of the former is the buffer overflow attack using the Unix "lpr" command in HP True64 Unix operating system as reported by CERT in CERT-VU #651377 (see http://www.kb.cert.org/vuls/id/IAFY-5DQPFL). We are more interested in addressing these types of attacks. The latter category is considerably more difficult to prevent, detect or deter. Addressing such threats is beyond the scope of the current work.

S. De Capitani di Vimercati et al. (Eds.): ESORICS 2005, LNCS 3679, pp. 231–246, 2005.

A second concern with intrusion detection systems is that they generate alerts only after they are able to see the misuse signatures or some deviations from norm. A malicious activity may result from a sequence of perfectly innocuous activities. Intrusion detection systems do not report on these activities mostly to prevent information overload for the system administrator. Thus the intrusion detection system generates an alarm only after the cause for alarm has occurred. In many situations however, this may already be too late.

These two factors lead us to propose a new approach that can be used to predict attacks arising from an insider's activities. Our work uses the user-intent analysis approach proposed earlier by Upadhyaya et al. [2, 14, 15, 16]. Upadhyaya et al's approach consists of ensuring that during a particular session a user remains reasonably within the scope of a previously declared set of activities. Any digression beyond this reasonable limit constitutes a misuse of system and steps are taken to protect against such digressions. However, this approach fails to account for the fact that a user may remain completely within the scope of a previosuly declared set of activities and still be able to launch attacks. This is where our approach contributes.

We begin by assuming that it is possible to enumerate the different attacks that a user can launch against a given system. This assumption is not unreasonable for known attacks. Almost all network vulnerability scanners provide such information. We then determine all the possible actions by which a user can launch an attack against the system. We map these actions against a user's session scope to identify which sequences of these actions can potentially lead to system compromise. Next we monitor each user's activities to see if and how they match against these sequences. Depending on the match we propose an estimator of attack probability.

Our approach is different from classical intrusion detection systems. It works as an early warning system. We continuously provide the system administrator an estimator of attack probability. Thus we cannot associate a rate of false positives or negatives with our technique. Our objective is to ensure that the system enters an alert mode once the probability of an attack is determined to be sufficiently strong. The notion of "sufficiently strong" is based on perceived risks. In the alert state, the following actions will be undertaken to ensure the survivability of information in case of an actual attack.

1. Allocate additional resources to assist in data collection by logging system wide activities more aggressively, saving system states more frequently and initiating recovery contingency plans by coordinating with other monitors.
2. Re-distribute essential services to other safer portions of the network.
3. Introduce mechanisms to handle possible attacks including ways to contain the attack.

All these activities are continued until either an intrusion is actually signaled by accompanying intrusion detection system or no further signs of attack are identified.

The advantage of our approach is that it is a flexible and resource efficient technique for security management. At the same time it is a guarded approach. If an attack succeeds (which is determined by techniques other than ours), it allows the system to be in a fully prepared mode for subsequent recovery.

The rest of the paper is organized as follows. In section 2 we briefly discuss the work by Upadhyaya et al. which is the starting point of our work. Section 3 discusses

our proposed approach. We begin the section with an overview of our approach. In section 3.1 we introduce the notion of augmented attack trees which helps us model system wide vulnerabilities that a user can potentially exploit. We further refine the augmented attack tree in section 3.2 to propose the minimal cut of an attack tree with respect to a given user intent. Finally section 4 concludes the paper.

2 Background and Related Work

In [14, 15, 16] the authors propose CIDS, a host-based concurrent intrusion detection scheme. The system is based on *user work profiling* [5]. This technique assumes that if one can encapsulate the intent of a user in a reasonable manner, then it is possible to assess intrusions by monitoring the activities on-line. The system works as follows. Sometime prior to login, a user submits a description of his intended system usage. This forms the user's *session scope*. The system converts the scope to a "SPRINT" (Signature Powered Revised Instruction Table) plan which is a list (may be ordered) of quadruples of the form $< \ subject, \ action, \ object, \ period \ >$. Here "subject" represents a user, "action" is an operation performed by the subject (such as login, logout, read etc.), "object" is the target of an action (such as files, programs, messages, printers etc.), and "period" represents the time interval for the duration of the action. Each quadruple represents a *verifiable assertion*, a concept that is a generalization of IDES's [4] specification of user characteristics, and can be monitored on-line. When a user is active, a monitor process (called the "Watchdog") monitors the user's commands and checks them against the user's SPRINT plan. Deviations beyond a certain tolerance limit is considered potential intrusions and CIDS generates alerts for such deviations. The basic CIDS system flow diagram is shown in figure 1.

The basic scheme [16] described above is improved upon by the authors in a later work [14]. In particular, the authors adopt the notion of reasonableness check to address

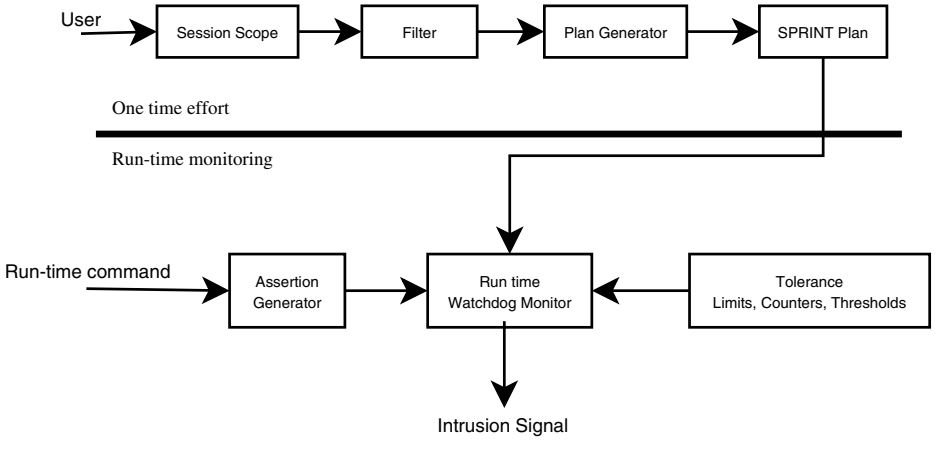

Fig. 1. Flow diagram for the CIDS system

issues like "what is a reasonable SPRINT plan" and "what is a reasonable deviation from the SPRINT plan". The authors use risk analysis techniques to estimate, from a given SPRINT plan, what events can possibly occur in the future (including their probabilities of happening) and the costs associated with those events.

One drawback of this work is that the authors do not address the scenario when a user does not deviate in any manner from the SPRINT plan, but still is able to launch an attack. The authors consider deviations from the SPRINT plan to be malicious, which is okay. However, their system fail to generate an alert if the user does not deviate from the SPRINT plan. Our work addresses this particular problem.

3 The Attack Prediction System

Our attack prediction system works briefly as follows. We begin by developing a model of network risks. We augment the notion of attack trees [8, 11] for this purpose. We introduce the notion of "attack probability" as labels of nodes in the attack tree. Next we iteratively apply each user's SPRINT plan to the augmented attack tree, to generate a trimmed attack tree for each user. We call such an attack tree the minimal cut of an attack tree with respect to the user intent. Branches of this trimmed attack tree represent, in a concise manner, all the different ways by which a user can use his assigned job privileges to launch an attack on the system. In the event such a trimmed attack tree does not exist for a particular user, we can safely claim that the user's current job description does not pose a threat to the system. This does not necessarily mean, however, that we can cease to monitor this user's activities. If we allow a user to deviate from her/his SPRINT plan as is done in the original work [2] then we should continue monitoring the user as proposed in that work. For this work we will assume that the users we are planning to monitor are the ones who, by virtue of their work definition, are able to launch attacks against the system.

In the following sections we describe each component of our system in details. We begin by describing how we augment the notion of attack trees to model network risks.

3.1 Augmented Attack Trees

Attack trees have been previously proposed [3, 8, 11] as a systematic method to specify system security based on varying attacks. They help organize intrusion and/or misuse scenarios by

1. utilizing known vulnerabilities and/or weak spots in the system, and
2. analyzing system dependencies and weak links and representing these dependencies in the form of an And-Or tree.

For every system that needs to be defended there is a different attack tree[1]. The nodes of the tree are used to represent the stages towards an attack. The root node of the tree represents the attacker's ultimate goal, namely, cause damage to the system. The

[1] Actually there can be a forest of trees. However, a forest can be always collapsed to a single tree. So we will assume that there is a single tree.

interior nodes, including leaf-nodes, represent possible system states (that is subgoals) during the execution of an attack. System states can include level of compromise by the attacker (such as successful access to a web page or successful acquisition of root privileges), configuration or state changes achieved on specific system components (such as implantation of Trojan Horses) and other sub-goals that will ultimately lead to the final goal (such as sequence of vulnerabilities exploited). Branches represent a change of state caused by one or more action taken by the attacker. Change in state is represented by either AND-branches or OR-branches. Nodes may be decomposed as

1. a sequence of events (attacks) all of which must be achieved for a this sub-goal to succeed; this is represented by the events being combined by AND branches at the node; or
2. a set of events (attacks), any one of which occurring will result in the sub-goal succeeding; this is represented by the events being combined by OR branches at the node.

The notion of attacks trees is related to the notion of attack graphs that have been proposed by other researchers [1, 6, 7, 9, 12, 13] for network vulnerability analysis. The difference is in the representation of states and actions. Attack graphs model systems vulnerabilities in terms of all possible sequence of attack operations. As pointed out by Ritchey and Ammann [10] a major shortcoming of this approach is its scalability. On the other hand, attack trees model system vulnerabilities in terms of cause and effect. Sequential ordering of events does not have to be captured in attack graphs. Thus constructing an attack tree is significantly less complex than attack graphs. An often cited criticism of attack trees (vis-a-vis attack graphs) is that they are not able to model cycles. However, we believe that this criticism is valid only in cases where attack trees are used to represent sequence of operations leading to attacks, not when they are used to represent the dependency of states reached. A second criticism of using attack tree to model attack scenarios is that they tend to get unwieldy. One contribution of this work is that we provide algorithms to minimize the size of the attck tree so that it is usable.

We assume that a technique is available to generate an attack tree corresponding to the network system we are attempting to defend. Figure 2 shows a simple attack tree for a hypothetical system that we are planning to defend.

In the network in figure 2 there is a server that stores and manages sensitive information. It is connected to a network printer. The network allows users to connect to systems either via wired connections or wirelessly. The ultimate objective of an attacker is to acquire root access on the server. To break the system the attacker may attack either via the buffer overflow attack on the lpr command or via the setuid command. These activities are represented as state transitions in the attack tree. Later on in the paper we define these activities as atomic attacks. The first atomic attack can be done without any pre-condition. The latter can be effected only if the attacker gains user privilege on the server machine.

We augment an attack tree by associating a label $< n, m >$ with each node in the attack tree. The augmented attack tree is defined formally as follows:

Definition 1. *An* augmented attack tree *is a rooted tree defined as* $AAT = (V, E, \epsilon, L)$*, where*

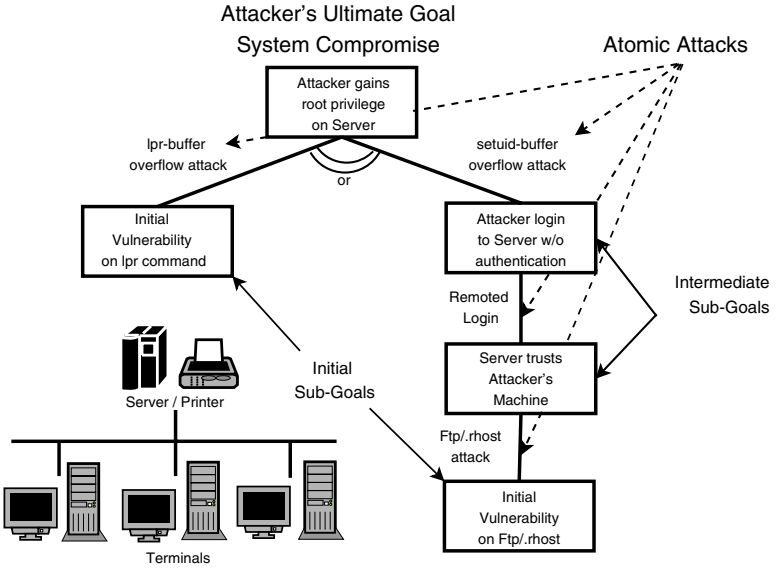

Fig. 2. Simple Attack Tree Corresponding to a Hypothetical System

1. *V is the set of nodes in the tree representing the different states of compromise or sub-goals that an attacker need to reach in order to compromise a system. $\mathcal{V} \in V$ is a special node, distinguished from others, that forms the root of the tree. It represents the ultimate goal of the attacker, namely system compromise. The set V can be partitioned into two subsets, $leaf\_nodes$ and $internal\_nodes$, such that*
   (a) $leaf\_nodes \cup internal\_nodes = V$,
   (b) $leaf\_nodes \cap internal\_nodes = \phi$, *and*
   (c) $\mathcal{V} \in internal\_nodes$
2. *$E \subseteq V \times V$ constitutes the set of edges in the attack tree. An edge $(v_i, v_j) \in E$ represents the state transition (in terms of actions taken) from a child node $v_i \in V$ to a parent node $v_j \in V$ in the tree. The edge (v_i, v_j) is said to be "emergent from" v_i and "incident to" v_j. Further if edges (v_i, v_j) and (v_i, v_k) exists in the set of edges, then v_j and v_k represent the same node.*
3. *ϵ is a set of tuples of the form $< v, decomposition >$ such that*
   (a) $v \in internal\_nodes$ *and*
   (b) $decomposition \in [AND - decomposition, OR - decomposition]$
4. *L is a set of attack probability labels. A label $l \in L$ is associated with a node. If $S \in V$ is a node then the attack probability label l_S, associated with node* S, *is given by the tuple $< n, m >$ where m and n are positive integers greater than 0 with $n \leq m$. The value of n for the node* S *can change over a period of time; however the value of m is fixed for the node* S. *The item m is termed the* least effort to compromise subgoal S *while the item n is termed the* number of currently compromised subgoals under S.

The values m and n in the attack probability label of the root node $\mathcal{V}$ are of particular interest to us. The ratio $\frac{n}{m}$ at any given time provides a measure of how far an attacker

has progressed towards the ultimate goal in terms of the least effort along the most advanced attack path that he has been through. Thus this ratio provides the probability of the system getting compromised at that time. The values m and n corresponding to the root node are computed based on the corresponding values for the other nodes. At this time we show how to compute the value of m for any given node. Note that this is a one time effort that is done during system initialization. First, some additional definitions.

Definition 2. *Given a node, v in an attack tree such that $v \in internal\_nodes$, the node is an* AND-decomposition *if all edges incident to the node are connected by the AND operation.*

Definition 3. *Given a node v of an attack tree such that $v \in internal\_nodes$, the node is an* OR-decomposition *if all edges incident to the node are connected by the OR operation.*

An AND-decomposition, v, (shown by a single arc among the edges incident to V in figure 2) means that each subgoal of v represented by a child of v needs to be reached in order to reach v. An OR-decomposition (shown by a double arc in figure 2) means that the goal v can be reach only if any one of the subgoals is reached. Note that reaching a child goal is only a necessary condition for reaching the parent goal and not a sufficient condition. An instance of an augmented attack tree is shown in figure 3.

Let us assume without loss of generality that the attacker uses one unit of *effort* to perform one atomic attack that furthers his goal. In other words, each hop along

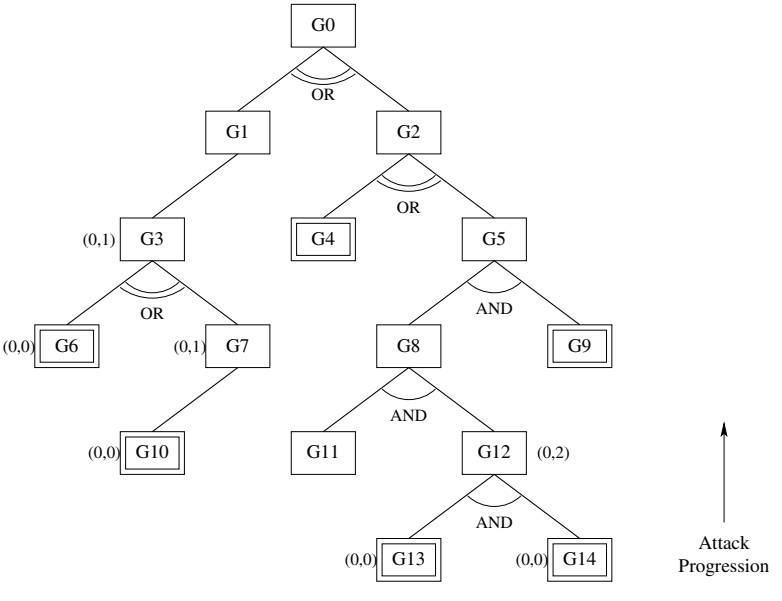

Fig. 3. Example of an Augmented Attack Tree

one edge of the attack tree takes one unit of effort to get through. The *least effort to compromise a subgoal* is the minimum effort the attacker needs to compromise the given subgoal. If a given goal *S* has an OR-decomposition, the least effort is computed as the minimum least efforts of its child nodes plus one unit effort needed to advance to *S* from the child node. If the goal *S* has an AND-decomposition then the least effort is the sum of the least efforts of the child nodes plus all additional unit efforts, one for each child node to go to *S*. The following definition captures the steps to compute the least effort for a subgoal *S*.

Definition 4. *Given a subgoal* S *and its child subgoals* S_i*, the* least effort to compromise S*,* m_s*, is defined as follows.*

1. *If* S *is a leaf node of the attack tree the least effort is 0.*
2. *If* S *is some interior node and is an AND-decomposition, then* $m_s = Sum(m_{s_i}) + k$ *where* k *is the number of child nodes* S_i *of* S.
3. *If* S *is some interior node and is an OR-decomposition, then* $m_s = Min(m_{s_i}) + 1$.

Henceforth we will use the terms attack tree and augmented attack tree interchangeably to mean the latter.

3.2 Minimal Cut of Attack Trees w.r.t. User Intent

An augmented attack tree can be used to model system vulnerabilities in a very effective manner. The attack tree describes all possible ways in which a particular attack can be launched. If there are more than one attacks against a system that we are concerned about, we can generate separate attack trees for each. However, there are a few drawbacks of the attack tree defined as it is now. First, for a complex system the attack tree can become quite deep and spread out. Thus it will become difficult to manage. Second, it is possible that a number of users are executing the same set of operations albeit at different paces. In this case, the cumulative effects of these users' actions will be reflected on the attack tree. If the users are not colluding this does not give the true picture of the state of the attack. For example, let a user have initially launched an attack and have compromised upto subgoal S_1 of S in an attack tree. Another user has compromised upto subgoal S_2. If the node S is an AND-decompostion of S_1 and S_2, the model will indicate that subgoal S is compromised. However, if the two users are not co-operating, then this is not the case. Thus, we want to refine the concept of attack tree so that we are able to monitor each individual user's activities. If we believe there is possibility of collusion among attackers we will maintain the system-wide attack tree as generated so far in addition to the per-user attack tree that we are now ready to define.

That a per-user attack tree is relevant is further strengthened by the following observation. For any attack, we may not always need to know all possible ways the attack can be launched, but rather the practical ways. In the case of attacks from insiders, for example, we are interested only in the activities of authorized users in the system. Thus, we want to determine if the operations that a user executes can lead to an attack. This implies that for a particular user, only a portion of an attack tree is relevant. This leads us to propose the notion of a minimal cut of an attack tree with respect to a user intent. We begin with the following definitions.

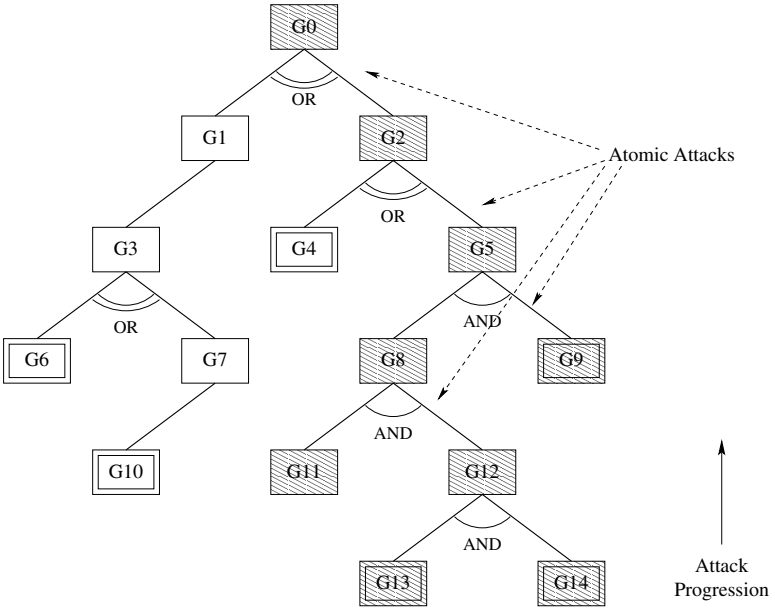

Fig. 4. A Possible Attack Scenario

Definition 5. *Given an attack tree, AAT, an* attack scenario*, AS of AAT is defined to be a sub-tree of AAT that is rooted at the root of AAT, and follows one or more branches through the tree to end at one or more leaf nodes of AAT such that*

1. *if the subtree has a node that is an AND-decomposition then the subtree must contain all the children of this node, and*
2. *the sub-tree represents one and only one of the many attacks described by AAT.*

The following figure represents one possible attack-scenario corresponding to the attack graph of figure 3, with the shaded boxes constituting the nodes in the attack scenario.

Definition 6. *An edge (v_i, v_j) in an attack scenario is called an* atomic attack*. The node v_i represents the precondition for the atomic attack and v_j is the goal.*

Referring to figure 4 some of the atomic attacks have been shown by dashed arrows. Note that to achieve an atomic attack, the attacker must execute some operations that exploit one or more vulnerabilities in the system. Once a vulnerability has been exploited the attacker executes a set of "attacking operations" that achieve the goal of an atomic attack. Thus,

Definition 7. *A suspicious operations set, SO^{attk}, corresponding to an atomic attack $attk$, is a set of operations on specific objects that may potentially lead to the culmination of the atomic attack $attk$. SO^{attk} is a set of tuples of the form $< action, object >$.*

We can identify two different types of operations in a suspicious operations set, SO^{attk}. The first subset of operations is the set Vul of vulnerable operations. At least one of the operations in the vulnerable set needs to be executed to exploit a vulnerability. An atomic attack can be launched by exploiting one or more vulnerabilities. Similarly each vulnerability can be exploited by executing one or more vulnerable operations. The second subset of operations is the set Ao of attacking operations. All of these needs to be executed to accomplish the atomic attack.

We would like to point out here that we specifically omit the use of the term "sequence" from the definition of suspicious operations set. It is quite possible that only a particular order of execution of the operations will lead to an attack. Since we are interested in estimating the probability of an attack and not just reporting on the attack if and when it is launched, we are interested in all the operations in the set and not just the operations in some particular order.

Definition 8. *The set of* intended operations *of a particular user, IOS, is a projection of the SPRINT plan for the user over attributes $action$ and $object$.*

We need to be worried about a user's SPRINT plan if some members of the corresponding intended operations set includes a suspicious operations set. We define the intended operations to *abet an attack subgoal* as follows.

Definition 9. *Given, an attack subgoal v_i (that is a node in an attack scenario) decomposed as the set A of atomic attacks $attk_i$, the suspicious operations sets corresponding to each atomic attack, SO_i^{attk}, and a set of intended operations, IOS for a user, we say that the intended operations* abet the attack subgoal *if and only if one of the following conditions holds true.*

1. *If v_i is an AND-decomposition with m edges then $\forall i,\ 1 \leq i \leq m, SO_i^{attk} \subseteq IOS$*
2. *If v_i is an OR-decomposition with m edges then $\exists i,\ 1 \leq i \leq m, SO_i^{attk} \subseteq IOS$*

The intended operations abet *an attack scenario if the intended operations abet all attack subgoal in that attack scenario.*

Recall that one of our objectives is to determine if a user's activities in a system can lead to an attack on the system. A related objective is to determine the exact way in which an attack can be launched with the user's intended operations. Thus, given an attack scenario, AS, consisting of subgoals (a_1, a_2, ..., a_n), we need to determine for each user's intended operations, if the intended operations abet every subgoal $a_i \in AS$. If the intended operations do not abet every subgoal a_i in AS, it implies that this particular attack scenario cannot arise from the user's activities. However, this does not mean that another attack scenario cannot arise from the same intended operations. What we need to identity, therefore, is the maximal set of attack scenarios that can arise from a given set of intended operations.

Definition 10. *The minimal cut, MC_{IOS}, of an attack tree, AAT with respect to a particular user intent, IOS, is the minimal subtree of AAT which is rooted at the root of AAT and whose leaf nodes are a subset of the leaf nodes of AAT, such that the subtree includes the maximal set and only the maximal set of attack scenarios that can arise from IOS.*

Algorithm 1 PRUNING ALGORITHM (AAT, r, IOS)

{**Description:** This algorithm takes an attack tree and a set of intended operations for a particular user and generates a confined version of the attack tree. The original attack tree is represented as a tree structure in which each node except the leaf nodes, contains an array of adjacency lists. Each element in the array represents an attack subgoal. For each attack subgoal, V_i, $Adj[V_i]$ contains a reference to a pre-condition subgoal. Each edge in the set E of edges refer to an atomic attack. The algorithm assumes the existence of a procedure called OPERATIONS that takes an edge $e[u, v]$ corresponding to a state transition in the attack tree and returns the set of operations that result in the state transition. It also assumes a second procedure called PARENT that takes a node v and returns the parent of v. The algorithm uses three temporary queues called Explore-List, E' and V' with operation ENQUEUE and DEQUEUE defined. Finally, the algorithm assumes a procedure DRAW_TREE that builds a tree given a set of nodes and edges}
{**Input:** The attack tree AAT, its root, r and the set of intended operations for the user, IOS}
{**Output:** The pruned attack tree containing the minimal cut of the attack tree with respect to the user intent.}

```
ENQUEUE(Explore-List, r)
E' ← φ
V' ← φ
while Explore-List ≠ φ do
   u ← DEQUEUE(Explore-List)
   ANY_MET ← false
   for all v ∈ Adj[u] do
      ENQUEUE(Explore-List,v)
      if OPERATIONS([u,v]) ⊆ IOS then
         ENQUEUE(E',[u,v])
         ANY_MET ← true
      end if
   end for
   if (Adj[u] = φ) && ([PARENT(u),u] ∈ E') then
      ENQUEUE(V',u)
   end if
   if ANY_MET = true then
      ENQUEUE(V',u)
   end if
end while
DRAW_TREE(V',E')
```

We now give two algorithms (algorithms 1 and 2) applying which in sequence gives us a minimal cut of an attack tree with respect to a given user intent. We call the first algorithm the Pruning Algorithm and the second algorithm the Trimming Algorithm. The first algorithm takes an attack tree and generates a subtree rooted at the root of the attack tree such that the subtree contains the desired minimal cut. It removes from the original tree any attack scenarios whose attack subgoals are not abetted by the intended operations. The second algorithm further reduces the subtree produced by the pruning algorithm to produce the minimal cut.

The following theorems hold on the pruning algorithm.

Theorem 1. *Let $ATT = (V, E, \epsilon, L)$ be an augmented attack tree and assume that the pruning algorithm is executed on ATT starting with the root node $r \in V$. If the user's actions abet an attack then that attack subgoal will be present in the pruned attack tree generated by the pruning algorithm.*

Proof. To prove soundness of the algorithm we need to prove that during its execution the pruning algorithm explores every state s that forms an attack subgoal for the attacker and includes it in the pruned attack tree. To prove completeness, we must prove that if a node s is included in the pruned attack tree it must form an attack subgoal for the attacker.

First let assume that at the termination of the pruning algorithm, an attack subgoal $s \in V$ which can lead to the root of the attack tree exists such that it is not enqueued by the pruning algorithm. The pruning algorithm starts by exploring the root's adjacent nodes and then iteratively explores the adjacent nodes of these. Thus, if there is an unexplored subgoal s left at the termination of the pruning algorithm, it must be the case that that subgoal s is not be reachable from the root.Then according to the definition 1, subgoal $s \notin V$ which contradicts the assumption.

We prove completeness of the algorithm as follows. Let us assume that at the termination of the algorithm there exist a state transition $e[u, v] \in E$ such that $SO_{e[u,v]} \subseteq IOs$ but $e[u, v] \notin E'$. Since all states in an attack tree have been explored, then $e[u, v]$ must have been explored. By definition 9, if user intent IOs abet an atomic attack which corresponds to the state transition $e[u, v]$, it must be explored and included in E'. This results in a contradiction.

The pruned attack tree generated by the pruning algorithm may include atomic attacks that however can never materialize from a user's activities. This is because the preconditions to these attacks are never satisfied by the user actions. For example, a user's intent may abet a remote login attack but may not abet the user to perform an ftp/.rhost attack on the target machine. In this case, the attacker cannot perform these atomic attacks at least till such time as they are not permitted to modify their intent. The next algorithm called the trimming algorithm removes these attack scenarios and produces the minimal cut of attack tree.

Theorem 2. *If the trimmed attack tree generated from a pruned attack tree by the application of a user's intended operation contains an attack scenario, then the intended operations abet that particular attack scenario.*

Proof. Let assume that there exist subgoal s (which its preconditions are met by the user intents) mistakenly removed by the trimming algorithm.

According to the semantic of the trimming algorithm, the *while loop* in the trimming algorithm explores every node in the input pruned tree and the *for loop* trying to discover all possible paths from the leaf nodes to a subgoal currently explored by the *while loop* 's iteration. Then the remove instruction removes a subgoal if and only if the previous *for loop* could not find such a path to the leaf node. We will split subgoal s in 2 cases.

Case 1: If subgoal $s \in V'$ is an initial subgoal, this case could not have happened since the if statement of *line 6* in the procedure detect the leaf node. Then the initial subgoal $s \notin$ Minimal cut if and only if $s \notin V'$ which contradicts the previous assumption.

Algorithm 2 TRIMMING ALGORITHM (V', E')

{**Description:** This algorithm takes the pruned attack tree generated by the pruninng algorithm, and removes attack goals that the user can never reach.}
{**Input:** The set of nodes from the pruned attack tree, ordered by traversing the tree in breadth first order and stored in an array V', and the corresponding set of edges E'}
{**Output:** Minimal cut of an attack tree with respect to user intent}

```
i ← SIZEOF(V')
while (i > 0) do
  u ← V'[i]
  i ← i − 1
  valid ← false
  if Adj[u] = φ then
    valid ← true
  else
    for all v ∈ Adj[u] do
      if v ∈ V' then
        valid ← true
      else
        remove (u,v) from E'
      end if
    end for
  end if
  if ¬ valid then
    remove u from V'
  end if
end while
DRAW_TREE(V',E')
```

Case 2: If subgoal $s \in V'$ is an intermediate subgoal, s will be removed by the trimming algorithm if and only if s can not be reached from any initial subgoal. This means the preconditions of an intermediate subgoal s are not met which contradicts the previous assumption.

3.3 Computing Probability of Attack by User

We now use the minimal cut of an attack tree with respect to a user intent to determine the probability of an attack originating from that user. Algorithm 3 computes the attack probability label of a subgoal at any given time t. By applying this algorithm on the root node of the minimal cut of an attack tree for a user, we get the attack probability label corresponding to the root at time t. The ratio n/m at time t gives the probability of the user's attack succeeding at time t.

Figure 5 shows an example trace for the algorithm 3. Assume that at time t a malicious user compromises the subgoals shown in the figure. The algorithm computes the $< n, m >$ value for the root of the tree as follows. All leaf-nodes return value (0, 0). Node A.1 and A.3 have the summation equal to (0, 0) since their immediate child nodes

Algorithm 3 Risk-Analysis(Subgoal A)

{**Description:** This algorithm takes an attack tree subgoal A and returns the attack probability label (n,m) for that goal. Here n refers to the number of nodes that have been compromised on the most advanced attack paths and m refer to the least-effort needed to compromise A on that path.}
{**Input:** A subgoal of an attack tree}
{**Output:** 1. Number of subgoals that have been compromised along the most advanced attack path. 2. Least-effort needed to compromise the subgoal along the most advanced attack path. }

Let n = number of currently compromised subgoal under A on the most advanced attacking path.
Let m = least-effort needed to compromise A on the most advanced path.
if A is a leaf node **then**
  return (0,0)
end if
if A is an AND-Decomposition **then**
  (n,m)← Sum {Risk-Analysis(A_i) | $\forall A_i Childnodesof A$}
  if A is compromised **then**
    return (n+k, m+k)
  else
    return (n, m+k)
  end if
else
  (n, m) ← Max{(n/m) of Risk-Analysis(A_i) | $\forall A_i Childnodesof A$}
  if A is compromised **then**
    return (n+1, m+1)
  else
    return (n, m+1)
  end if
end if

are all leaf nodes. When A.1 is compromised the procedure returns (2, 2). Similarly for A.3 the procedure returns (0, 2). For B.1 and B.3 the values are (1, 1) and (0, 1) respectively. At this point the value of (2, 2) tells us that it takes 2 unit efforts to compromise A.1. The attacker has already compromised A.1 but no damage has been done on A.3. Next we calculate value on A. Eventually, since the root is an AND-decomposition on two branches A and B the least effort is 8 + 2 = 10 and the number of compromised nodes is 3. This yields a probability of attack value of $\frac{3}{10}$ at time t.

4 Conclusions and Future Work

In this paper, we propose a proactive approach to predicting network attacks that can potentially result from a insider exploiting known system vulnerabilities through the execution of authorized operations. Our system is intended to complement existing intrusion detection systems to help fight unwelcome cracker activities. We develop a quantitative framework. Our approach is proactive in the sense that we want to provide the system administrator an early warning. We want the system to enter an alert mode

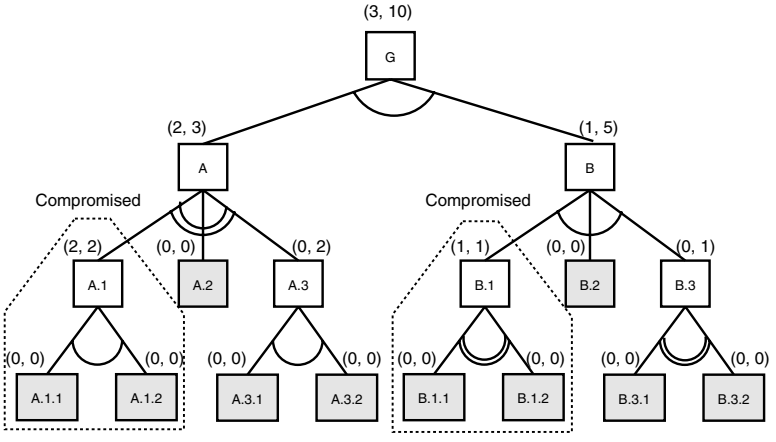

Fig. 5. Computing attack probability label for attack subgoal

once the probability of an attack is determined to be sufficiently strong and remain in that state until either an intrusion is actually signaled by an intrusion detection system or no further signs of attack are identified. The advantage of our approach is that it is a flexible and resource efficient technique for security management. At the same time it is a guarded approach. If an attack succeeds, it allows the system to be in a fully prepared mode for subsequent recovery. Moreover, although we develop this framework with insider threats in mind our approach can be easily adapted to predict threats from the outside.

At this stage the model has one shortcoming. The model results in the probability of an ensuing attack to increase continuously or remain static at best. However, in real life, a system administrator may sometimes take certain steps that results in the signs of an attack to subside. In our model this will be reflected by no change in the probability of the attack. We are currently working on this problem. Once that is solved we plan to develop a working prototype and undertake some simulation studies.

References

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington DC.*, pages 217–224, November 2002.

[2] R. Chinchani, S. Upadhyaya, and K. Kwiat. Towards the scalable implementation of a user level anomaly detection system. In *Proceedings of the 2002 IEEE MILCOM Conference, Volume 2*, pages 7 – 10, Anaheim, CA, October 2002.

[3] J. Dawkins, C. Campbell, and J. Hale. Modeling network attacks: Extending the attack tree paradigm. In *Proceedings of the Workshop on Statistical Machine Learning Techniques in Computer Intrusion Detection, Baltimore, MD*. Johns Hopkins University, June 2002.

[4] D. Denning and P. Neumann. Requirements and model for "ides" - "a" real-time intrusion detection expert system. Technical report, Technical Report, Computer Science Laboratory, SRI International, 1985.

[5] The SANS Institute. Intrusion detection faq. Available at `http://www.sans.org/resources/idfaq`, April 2004.
[6] S. Jha, O. Sheyner, and J. Wing. Minimization and reliability analysis of attack graphs. Technical Report CMU-CS-02-109, School of Computer Science, Carnegie Mellon University, February 2002.
[7] S. Jha, O. Sheyner, and J. Wing. Two formal analyses of attack graphs. In *Proceedings of the 2002 Computer Security Foundations Workshop, Nova Scotia*, pages 45–59, June 2002.
[8] A.P. Moore, R.J. Ellison, and R.C. Linger. Attack modeling for information survivability. Technical Note CMU/SEI-2001-TN-001, Carnegie Melon University / Software Engineering Institute, March 2001.
[9] C. Phillips and L.P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 New Security Paradigms Workshop, Chicago, IL*, pages 71–79, January 1998.
[10] R. W. Ritchie and P. Ammann. Using model checking to analyze network. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.
[11] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, December 1999.
[12] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Computer Society Symposium on Security and Privacy, Oakland, CA*, May 2002.
[13] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *Proceedings of DISCEX '0: DARPA Information Survivability Conference and Exposition II*, pages 307–321, June 2001.
[14] S. Upadhyaya, R. Chinchani, and K. Kwiat. An analytical framework for reasoning about instrusions. In *Proceedings of the 2001 IEEE Symposium on Reliable Distributed Systems*, pages 99–108, New Orleans, LA, October 2001.
[15] S. Upadhyaya, R. Chinchani, and K. Kwiat. A comprehensive reasoning framework for information surviability. In *Proceedings of the 2nd Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop*, pages 148–155, West Point, NY, June 2001.
[16] S. Upadhyaya and K. Kwiat. A distributed concurrent intrusion detection scheme based on assertions. In *Proceedings of the SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems, Chicago, IL*, pages 369–376, July 1999.