# Plan Recognition in Intrusion Detection Systems

Christopher W. Geib and Robert P. Goldman
Honeywell Labs
3660 Technology Drive
Minneapolis, MN 55418 USA
{geib,goldman}@htc.honeywell.com

## Abstract

*To be effective, current intrusion detection systems (IDSs) must incorporate artificial intelligence methods for plan recognition. Plan recognition is critical both to predicting the future actions of attackers and planning appropriate responses to their actions. However network security places a new set of requirements on plan recognition. In this paper we present an argument for including plan recognition in IDSs and an algorithm for conducting plan recognition that meets the needs of the network security domain.*

## 1. Introduction

Intrusion detection systems (IDSs) must move from describing actions that have already happened to predicting future actions. For IDSs to move forward, they must be able to analyze the actions of a hacker[1], infer the hacker's goals, and make predictions about his/her future actions. In the artificial intelligence (AI) literature this process of deducing an agent's goals from observed actions is called plan recognition or task tracking. We argue that plan recognition must be a central component in future IDSs.

However, most existing AI literature on intent recognition makes a number of assumptions preventing its application to the computer network security domain. In our previous work[8] we have described an approach to plan recognition that does not make the restrictive assumptions of other AI intent recognition systems. In this paper we discuss its application to the network security domain.

Other work in network security has argued for network level coordination among IDSs[1, 6] and even referenced inferring attacker intent [10] as a motivation for this. However, these papers have focused on the protocols and communication issues surrounding this kind of distributed coordination. In contrast, this paper focuses on the need for and application of AI research in plan recognition to the problems of network security and the inference of attacker intent.

The remainder of this paper has the following structure. First, we will argue that plan recognition is a crucial addition to network security work. Second we will provide an overview of previous work in plan recognition. Third we will describe our implemented theory of plan recognition for hostile agents for a computer network security domain. After presenting the formalization, we provide an example, showing how the theory is used and how it differs from previous approaches. Then we conclude with remarks and plans for future work.

## 2. The Need for Plan Recognition

Current IDSs do not predict attacks; they do not provide an early warning. They report the type and properties of an attack after it has happened. As such these systems are often reduced to the role of *post mortem* analysis rather than being proactive. While recognition of attacks is an important ability, it falls short of the community's vision for IDSs as systems that predict future hacker actions and automatically and correctly respond to attacks in a timely manner.

To be proactive IDSs must be able to infer the goals of attackers. Identifying the attacks is not sufficient. To see this, consider the case of an IDS report of a synflood. For the purposes of this example assume that the attacker it using this synflood for one of two reasons.

1. a denial of service(DOS) attack to prevent our use of the machine.
2. Suppressing a host during an IP spoofing attack on another machine.

To correctly respond to this attack an IDS needs to understand the intent of the attacker, predict the next actions

[1] We apologize for the use of the term "hacker" in its criminal sense, but we will use this as a convenient short hand in this paper

of the attacker and then take actions to prevent these future actions. To see this, consider each of the possible intents in turn.

First, suppose the attacker is using the synflood to prevent our use of the machine. Knowing this, we would predict that the attacker's future actions will be to continue the flood of SYN packets to suppress the machine. To respond to this attack we can modify the firewall to reject packets from the attacking host or to only allow a specified number of connections from the attacking host to the DOSed machine. This is will effectively limit the number of open connections and prevent the DOS attack.

Second, suppose the attacker is using the synflood as part of an IP spoofing attack and his goal is access to a different machine entirely. Knowing this we would predict that we would see packets that appear to originate from the suppressed host, we might see the synflood stop on its own and possibly the establishment of a connection from outside the network to another host on the network (one the original host trusted). To respond to this attack we should modify the firewall to prevent all external connections to all machines that trust the synflooded machine.

Notice that while the reported action is the same, the correct response is completely different. In fact the response in the first case will have no effect if the attacker's goal is access to another machine. By the time the firewall has been modified and the synflood clears the IP spoofing attack will have been executed and the attacker will likely already have access to the machine. Conversely if the attackers real intent is just to DOS the selected host, responding as though an IP spoofing attack is underway will cut off connections to other machines from the internet. In short, inferring that the attacker's goal is "access to the system" rather than "denial of services" is critical in both making predictions about what the attacker will do next and taking the correct countermeasures.

What we are suggesting is that IDSs need to combine multiple reports and information to identify the attacker's goals. In this case, there is a convenient clue to the attacker's intent: part of an IP spoofing attack is the sending of packets to a host with the source IP address of the DOSed host. Given a network based IDS that can watch for these anomalous packets, recognizing the synflood as a DOS versus part of an IP spoof is relatively easy. Conversely if no spoofed packet is observed the attacker is likely engaging in a simple DOS.

This example illustrates the kind of reasoning that we are advocating. By taking the output reports of current IDSs as a stream of observed actions, and using intent recognition techniques, it is possible to infer the attacker's goals and thus accurately direct responses. However this simple description paints too rosy a picture. Previous work on intent recognition has made a number of simplifying assumptions that would prevent its application to this domain. In the following section we will discuss these assumptions and the requirements placed on plan recognition systems by the network security domain.

## 3. Requirements on Plan Recognition

The requirements that are placed on our plan recognition system come from two different aspects of the network intrusion detection problem. First, we are attempting to infer the plans of covert agents. As we already know hackers often take deliberate actions to "cover their tracks" and hide their actions and intentions. This will place significant requirements on the process of plan recognition that are not true when the agent being observed is cooperative.

Second, taking plan recognition in the computer security domain seriously requires confronting a number of issues that have not been examined in more theoretical or academic domains. In this case, the plans the hackers are following have properties that are not as prevalent in the domains that have been the traditional areas for plan recognition. In the following sections we will consider the requirements placed on an effective plan recognition system by these factors in turn. In each case we will identify the requirement and attempt to provide a motivating example for it.

### 3.1. Hostile agents

Most previous work in plan recognition has assumed cooperative agents. This domain makes this assumption untenable. We have identified the two significant requirements that hostile agents place on plan recognition. They are the ability to infer unobserved actions from observed actions and inferring unobserved actions from observations of state change We discuss these in turn.

**Unobserved actions:** Given that hackers may be using new exploits, it is entirely possible that they may have actions that our current IDSs do not recognize. Consider a conventional signature detector when faced with an unknown exploit. Since it doesn't have the signature for the attack it will not report. In some cases, even small variations of an exploit can make an attack invisible to a signature detector.

Further in real networks there are often "holes" in the IDSs coverage. That is, hosts that do not have sufficient sensor coverage to detect all of the malicious activities that might occur on the system. Hackers entering a system through one of these sensor holes will not be observed by the system's IDSs.

If our plan recognition system is to be successful in this domain, it must be able to infer the occurrence of actions that it has no report of when other evidence suggests they

have occurred. Consider the case of an attack on a single machine that is observed without any preliminary scanning. Since we know that identifying the IP address and relevant port numbers are important for this attack, we can infer that some scanning or information-gathering action must have occurred before the attack, even though we did not observe it.

**Observations of state changes:** Consider the report of a new service running on a host. Note that we distinguish between a report of the action of starting the service and a report that the service is now running and it previously was not. The first report would be generated by a host based IDS that watched the hacker start the service. The report of a state change might be generated by a network-based IDS that scans to identify that no unauthorized services are being run. In the first case we see the action and in the second we only observe the effect of the action; we receive a report of a state change. From this state change we can infer that there was an action that caused it.

Existing work has not examined the issue of reports of state changes. In the case of cooperative agents there is no need. If the agent is cooperative we can assume we have a complete list of the agent's actions. There is no reason to infer the execution of unobserved actions from state changes; the complete set of actions is already available. However with an incomplete record of the agent's actions, reports of state change can provide evidence of unobserved actions.

### 3.2. Real World Computer Security

For our plan recognition algorithms to be effective in deployable IDSs, we must enforce a number of requirements on our system. These requirements include the ability to reason about: partially ordered plans, multiple concurrent goals, actions used for multiple effects, failing to observe an action, the effect of world state on the attackers plans, and multiple possible hypotheses. We consider each of these in turn.

**Partially ordered plans:** The plans hackers follow are often very flexible in the ordering of their plans steps. Consider system scanning by IP-sweeping and port scanning. These steps can be interleaved in at least two orders.

1. Collect a large number of IP addresses and then port sweeping each of them.
2. Port sweep each IP address as it is found.

While port sweeping a host can only be done after the host's IP address has been identified, there are no other constraints on the order of the actions. This means that thee port sweep actions for a subdomain are not ordered with respect to each other but only with respect to the IP address discovery process. In short, the port sweep actions can be executed in many acceptable orderings.

In the AI planning and plan recognition literature, plans that have this kind of flexible ordering between the plan steps are called *partially ordered plans*. In these cases, the ordering constraints of the plan only establish a partial order over the actions of the plan. In contrast when the ordering constraints impose a total order on the actions of the plan we call this a *totally ordered* plan.

Since the plans that are followed by the attackers have this more flexible, partially ordered structure, we will require that our system be able to recognize the multiple possible instantiation orderings created by these plans.

**Multiple concurrent goals:** Hackers often have multiple goals. That is, a hacker might be interested in stealing your sensitive corporate data *as well as* using your computers to launch attacks against other targets. Much previous work in plan recognition has looked for the single goal that best explains all the observations. In contrast we will require that our plan recognition system be able to consider cases where the agent has multiple goals.

**Actions used for multiple effects:** Often in the computer security domain a single action can be used for multiple effects. Consider the scanning of a subdomain. This information can be used both for a DOS attack as well as to identify the web server that hacker wants to deface. In this case, it is not necessary for the attacker to perform this same scan for each goal; they can do it once for both. In effect they "overload" the scanning action and use it to contribute to multiple goals. A critical requirement for our plan recognition system is that it be able to handle these kinds of actions.

**Failure to observe:** Suppose we observe a scanning of our subnetwork. The longer the we go without seeing any further activity, the more likely we are to believe that this was just an isolated scanning event. It was not part of a larger plan. However, if right after the scanning event, we see other malicious activity then we are more likely to believe the scan is the reconnaissance step of a plan.

In this case, since we are expecting to see malicious activity following the scan, when we don't see it, we change our belief in the likelihood that the scan was part of an attack and instead attribute it to a "random" scan. More formally, the failure to observe actions that confirm our hypothesis results in lowering our estimate of how likely we think the hypothesis is.

Consider the case of an IDS that is 99% effective at identifying a particular exploit. On the basis of this very reliable detector we can make a number of inferences. One of the most important is that if we don't receive a report from the detector then it is very unlikely that the attacker has executed this exploit. In general, there are a significant number of conclusions that one can draw from the failure to observe actions and we will require that our system be able to perform this kind of reasoning.

Our previous commitment to considering agents with multiple concurrent goals makes it even more critical that our plan recognition system be able to engage in this kind of reasoning. It is rare that we will be provided with definitive evidence that an attacker is *not* pursing a specific goal. Far more likely is that a lack of evidence for the goal will lower its probability. As a result, reasoning on the basis of the "failure to observe" is critical for a plan recognition system to prefer those explanations where the agent is pursuing a single goal over those where the attacker has multiple goals but has not performed any of the actions for one of them.

**Impact of world state on adopted plans:** World state can have significant impact on the goals that are adopted by an attacker. Consider the case of a computer network security firm that has not locked down its public web-server outside its firewall. When an attacker sees this we would hardly find it surprising if they adopt the goal of defacing the firm's web-page. In general, situational factors can have a significant effect on the goals adopted by agents in any real world domain. We will require our plan recognition algorithm be able to handle these effects.

**Consideration of multiple possible hypotheses:** Providing a single explanation for the observed actions in general is not going to be as helpful as ranking the possibilities. Consider the case where all we observe is scanning activity. While this indicates a hacker is interested in our network, by itself it provides very little evidence about the hacker's intent. Rather than giving just one of the many equally likely answers it is much more helpful to report the relative likelihood of each of the possibilities. This provides the information that there are multiple equally likely hypotheses to explain the observations rather than a single most likely one.

From here on, our discussion of issues and solutions will be helped by a specific motivating example. Therefore the following section will provide a brief introduction to our plan representation and an example plan library that we will use for the remainder of the paper. Following this introductory material we will discuss some background information on the existing AI work in plan recognition and where it has met the requirements we have specified and where it has failed.

## 4. Plans

In this paper, we use simple hierarchical (task decomposition) plans[5], as most plan recognition work does. We assume that agents have a *plan library* that provides recipes for achieving goals. Figure 1 shows a plan library for a "hacker" in a simplified computer network intrusion example.

If a hacker has a goal like stealing information from a computer (**theft**), the plan library breaks that goal into five *steps*: scan the system to determine vulnerabilities (**recon**), exploit the system's weaknesses to gain entry (**break-in**), escalate privileges (**gain-root**), export desired data (**steal**), and hide traces of presence on computer (**clean**). Ordering constraints within a method are represented by directed arcs. For example, the hacker must **break-in** before she can **gain-root**.

Finally, notice that there is a condition/event that is tied to the action **clean**. The dashed line represents the fact that this condition results from the execution of the action. Thus, if **clean** is executed it will result in deleted event logs (**deleted-logs**). This information about action effects will be critical to inferring the execution of unobserved actions.

## 5. Plan recognition background

Plan recognition is the process of inferring the goals of an agent from observations of an agent's actions. Cohen, Perrault and Allen [4] distinguish between two kinds of plan recognition, *keyhole* and *intended* plan recognition. In keyhole recognition, the recognizer is simply watching normal actions of an agent. The agent does not care or is not aware that their actions are being observed. They are simply engaging in the task. In intended recognition, the agent is cooperative; its actions are done with the intent that they be understood. This may result in the agent performing the action in a particular or stylized way in an effort to assist the recognizer in the task. Intended recognition arises, for example, in cooperative problem-solving and in understanding indirect speech acts. In these cases, recognizing the intentions of the agent allows us to provide assistance or respond appropriately.

From these two kinds we distinguish *adversarial* plan recognition. It arises in contexts like network security, military intelligence and game-playing, where the agent is actively hostile to the observation of their actions and the inference of their plans. As we have pointed out already, adversarial plan recognition requires the violation of a number of assumptions that are reasonable in the cases of keyhole and intended plan recognition.

The earliest work in plan recognition (e.g., [16, 20]) was rule-based; researchers attempted to come up with inference rules that would capture the nature of plan recognition. To the best of our knowledge, Charniak [3] was the first to argue that plan recognition was best understood as a specific form of the general problem of *abduction*, or reasoning to the best explanation. Abduction, as opposed to deduction or induction, is reasoning from "A implies B" and knowledge of "B" to deduce "A". This is the reasoning pattern for most kinds of diagnosis.

In 1986, Kautz and Allen (K&A) published "Generalized Plan Recognition," [13]. This work has framed almost all subsequent work in plan recognition. K&A defined the

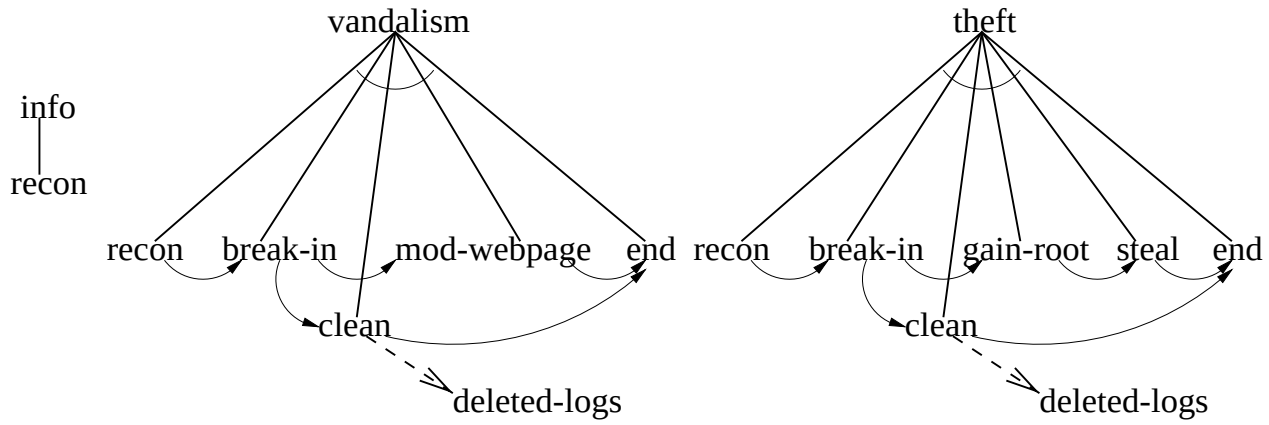


**Figure 1. A hierarchical plan library in diagram form.**

problem as the problem of identifying a minimal set of *top-level actions* sufficient to explain the set of observed actions.

Plans were represented in a plan graph, with top-level actions as root nodes and other actions as nodes implying the top-level actions. To a first approximation, the problem of plan recognition was then a problem of graph covering. They treated the problem as one of computing minimal explanations, in the form of vertex covers, of the plan graph. They formalized this in terms of McCarthy's circumscription.

For example, if one observed **recon** (See Figure 1) the three minimal explanations would be:

$$(\textbf{theft}) \vee (\textbf{vandalism}) \vee (\textbf{info})$$

Notice that, with only this observation we have no evidence to rule out the possibility that the agent has multiple goals. This single action could also explain *two* or even *three* top-level goals like:

$$(\textbf{theft} \wedge \textbf{vandalism})$$

or

$$(\textbf{theft} \wedge \textbf{vandalism} \wedge \textbf{info})$$

K&A insistence on a minimal set of top-level actions prevents the consideration of these possibilities. Even if they are equally likely. This violates our requirement of the ability to support multiple concurrent goals and the consideration of multiple possible hypotheses.

Another problem for the use of K&A's approach in the network security domain is that it does not take into account differences in the *a priori* likelihood of different plans. Charniak and Goldman (C&G) [2] argued that, since plan recognition involves abduction, it could best be done as probabilistic (Bayesian) inference. Bayesian inference supports the preference for minimal explanations in the case of hypotheses that are equally likely (as in the previous case.) It also correctly handles explanations of the same complexity but different likelihoods. For example, it is possible for a legitimate user to add a .rhosts file to a long dormant account, but it is far more likely that we have had an intrusion.

Two plan recognition situations that are not handled by either K&A or C&G are the problems of influences from the state of the world and evidence from failure to observe actions. As we discuss in Section 3 the state of the world will influence an agent's decision to pursue plans. K&A could not take this into account, because they did not consider the relative likelihood of plans. Even for C&G, it is not simple to take this into account because they defined their probability distributions over the plan library.

The problem of evidence from failure to observe is a more complex one. Consider what would happen if one observed **recon** and **break-in**. Assuming that they were equally likely a priori, one would conclude that either **theft** or **vandalism** were equally good explanations (see Figure 1). However, as time went by and one saw other actions, without seeing **mod-webpage**, one would become more and more certain that **theft** was the right explanation. Systems like those of C&G and K&A, are not capable of reasoning like this, because they do not consider plan recognition as a problem that evolves over time. They cannot represent the fact that an action has not been observed *yet*. They can only be silent about whether an action has occurred — which just means that the system has failed to notice the action, not that the action hasn't occurred — or assert that an action has not *and will not* occur.

Vilain [18] presented a theory of plan recognition as parsing, based on K&A's theory.[2] Vilain does not actually propose parsing as a solution to the plan recognition prob-

[2] This was not the first attempt to cast plan recognition as parsing [17].

lem. Instead, he uses the reduction of limited cases of plan recognition to parsing in order to investigate the complexity of K&A's theory. The major problem with parsing as a model of plan recognition is that it does not treat partially-ordered plans or interleaved plans well. Indeed, partial ordering (**clean** and **gain-root** can be done in any order, as long as **break-in** is done first), would cause an explosion in the size of Vilain's grammars.

There are grammatical formalisms that are powerful enough to capture interleaving. The central advantage of parsing as a model is that it admits of efficient implementation when restricted to context-free languages. There are context-free parsing algorithms that are $O(n^3)$ which would make for very efficient plan recognition, however, if we increase the power of the grammar to admit interleaved plans, these efficient algorithms are no longer available to us.

More recently, Wellman and Pynadath (W&P) [19] have proposed a plan recognition method that is both probabilistic and based on parsing. W&P represent plan libraries as probabilistic context-free grammars (PCFGs) and extract Bayes networks from the PCFGs to interpret observation sequences.

Unfortunately, this approach suffers from the same limitations on plan interleaving as Vilain's. W&P propose that probabilistic context-*sensitive* grammars (PCSGs) might overcome this problem, but it is difficult to define a probability distribution for a PCSG [15].

Huber, et. al. [11] present an approach to keyhole plan recognition for coordinating teams of agents based on the Procedural Reasoning System(PRS)[7, 12]. PRS is a planning architecture that uses hierarchical plan specifications very similar to our plan library and a reactive execution engine to allow the system designer to build agents that follow the specified plans.

Huber's algorithm automatically generates plan recognition belief networks from PRS plan specifications. The most important difference between our work and theirs is that we obtain a simpler structure by working with the plan representation directly, instead of generating a belief network as an intermediate representation. Further, it is not clear how they handle the interleaving of multiple plans and the development of plans over time.

In the following section we will describe our previous work (GG&M)[8] on plan recognition and some additions to it that were made for the network security domain. The central motivations for our previous work are the same shortcomings in previous plan recognition systems we have pointed out in this section namely:

- partially-ordered plans and plan interleaving;
- multiple concurrent goals;
- actions used for multiple effects;
- evidence from the failure to observe expected actions;
- contextual influence on plan choice;
- consideration of multiple possible hypotheses

In the following section, we will provide a brief overview of our system that handles these concerns and then turn to a discussion of how to handle the issues raised by adversarial plan recognition.

## 6. Recognition based on execution

The plan recognition framework developed in GG&M is based on the realization that plans are executed dynamically and that at any given moment the agent is able to choose to execute any of the actions that have been enabled by its previous actions. Thus, at any time an agent will have a *pending set* of actions that are enabled by its previous actions. The agent is free to choose to execute any of the actions in the current pending set.

To formalize this slightly, initially the executing agent has a set of goals and chooses a set of plans to execute to achieve these goals. The set of plans chosen determines the set of *pending* primitive actions. As the episode proceeds, the agent will repeatedly execute one of the pending actions, and generate a new set of pending actions from which further actions will be chosen.

The new pending set is generated from the previous set by removing the action just executed and adding newly enabled actions. Actions become enabled when their required predecessors are completed. This process is illustrated in Figure 2. To provide some intuition, the sequence of pending sets can be seen as a Markov chain, and the addition of the action executions with unobserved actions makes it a hidden Markov model.

To use this model to perform probabilistic plan *recognition*, we use the observations of the agent's actions as an execution trace. By stepping forward through the trace, and hypothesizing goals the agent may have, we can generate the agent's resulting pending sets. Once we have reached the end of the execution trace we will have the complete set of pending sets that are consistent with the observed actions and the sets of hypothesized goals that go with each of these sets. Once we have this set we establish a probability distribution over it. We can then determine which of the possible goals the agent is most likely pursuing.

Notice that the observations of the agent's actions are used to construct the execution traces. In the case of hostile agents, the observations will not, in general, be a complete record of the execution trace. Instead it will be necessary to consider execution traces containing unobserved actions.

This theory was designed to handle: partially ordered actions, overloaded actions, the effects of context, and negative evidence from not observing actions (i.e. the dog didn't

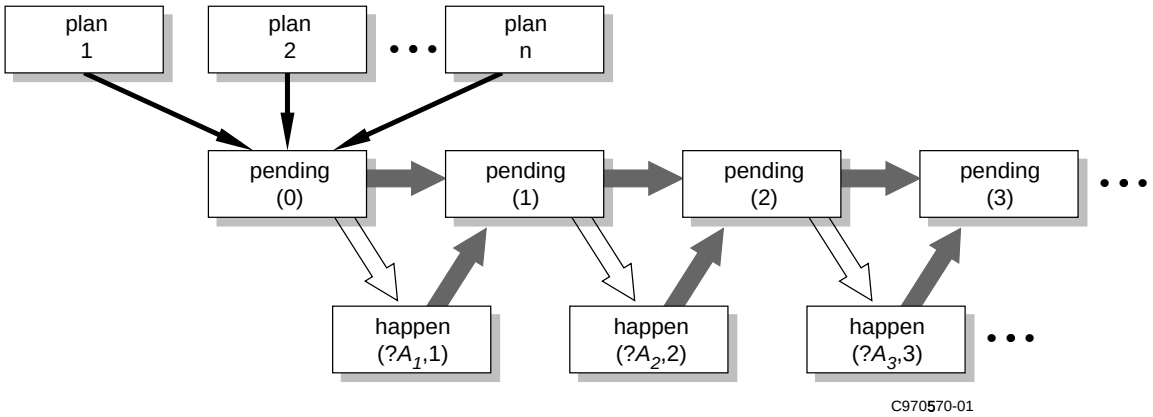


**Figure 2. Generation of pending sets.**

bark). While some of these problems are partially handled by other systems, no other system handles all of them. We refer the reader to GG&M for a complete discussion of this formalism. We will now consider extending this formalism to the problems presented by hostile agents.

## 7. Problems with hostile agents

Existing work on plan recognition has assumed complete observability of the agent's actions. Taking adversarial plan recognition seriously means that we can no longer rely on this. That is, we want to infer the goals of an agent given that the behavior of the agent is only partially observable. Earlier we pointed out requirements on systems that want to move away from this assumption. They must be able to infer unobserved actions from observed actions, and they must be able to infer unobserved actions from state changes.

The rest of this paper will be organized as follows, first we will discuss how we have added these two kinds of reasoning to our system. We then discuss our general algorithm for plan inference, and we will conclude with a discussion of the assumptions and limitations of the algorithm.

### 7.1. Inferring unobserved actions from observed actions

Consider the following observations:

(**gain-root**,**mod-webpage**)

These two observations indicate with very high probability that the hacker is engaged in both stealing information from a computer and defacing a webpage. We can conclude this because these actions are members of disjoint plans, that is, no single root goal will explain both of these actions.

These actions are even more informative since they are both *unenabled* by the observed actions. We define an *unenabled action* is one that is observed without having first observed the actions the plan library specifies must come before it. In this case, the plan library specifies that **recon** and **break-in** must occur before **gain-root** or **mod-webpage**. Therefore, in order to explain these two observations we must assume the execution of at least one instance of **recon** and **break-in** each. Thus, these two actions provide evidence of two distinct plans:

(**recon**, **break-in**, **mod-webpage**)
and
(**recon**, **break-in**, **gain-root**)

Consider our model of plan recognition. Unenabled actions provide more information for us to use to reconstruct the agent's actual actions than other observations. They require that the action itself be in the sequence, but they also provide evidence of unobserved actions. Consider generating the execution traces needed to produce the pending sets for the last example. Not only does this set of observations allow us to prune out any execution sequence that doesn't contain a **gain-root**, followed sometime later by a **mod-webpage**, but it also allows us to ignore any trace that doesn't have a **recon** followed by a **break-in** preceding the **gain-root**. These unenabled actions are very important pieces of information when attempting to infer the plans of hostile agents.

Note that in this discussion, we have implicitly assumed the agent can perform *any* action without detection, however, in practice this is not true. Some actions are simply harder to hide than others. For example, the probability that a person could conduct a port scan of my machine without my knowledge is much higher than the probability that they could successfully carry out a denial of service attack against it without my noticing. In this framework it is trivial to add probabilities about the likelihood of an agent performing a specific action undetected.

### 7.2. Inferring unobserved actions from state changes

Often, when it is possible to prevent an observer from seeing the performance of an action, it is **not** possible to prevent the observation of the action's effects. In our network security domain consider the **clean** action; the execution of the action might be hidden, but the deleting the log files is very visible.

Reports of state changes can provide evidence of unobserved actions that have the desired effect. From them we can infer that the action has occurred before the report of the state change. Reports of state change can also provide confirming information about a previously observed action.

Consider the following sequence of observations:

(**recon**,**break-in**,**deleted-logs**)

The report of the deleted event logs implies an unobserved **clean** action. Further the ordering constraints in the plan library imply that it must fall between the execution of **break-in** and the report of **deleted-logs**. However, if the sequence of observations were:

(**recon**, **break-in**, **clean**, **deleted-logs**)

The report would provide no extra information since it is consistent with the observed actions. Like acquiring evidence from unenabled actions these reports give more information about the execution traces that are consistent with the observation.

### 7.3. The solution

The central idea behind our plan recognition algorithm is the production of a probability distribution over the set of all pending sets. This is generated using the observations as an execution trace of the agent's actions. Since each pending set is used in at least one execution trace, we generated the pending sets by stepping through observations. In the case of cooperative agents with complete and correct observations, this is sufficient.

In contrast, in the case of hostile agents we face a problem with the execution traces. We can no longer assume that the observation stream is complete; it no longer represents the complete execution trace. Instead, for each set of observations we must construct the *set* of possible execution traces, inserting hypothesized unobserved actions to complete them.

For easy implementation we have assumed a bound on the number of unobserved actions. The next section discusses removing this assumption. Given a finite set of primitive actions, bounding the number of unobserved actions provides a limit on the length and number of execution traces that must be considered. In the worst case we only need to consider all execution traces whose length is equal to the maximum number of unobserved actions plus the number of observed actions. This sounds like a very large search space, however, we can prune this set of execution traces with the ordering constraints provided by the observations.

We are only interested in execution traces consistent with the observations, therefore if a sequence does not contain all the observed actions or doesn't obey the ordering constraints imposed by the sequence or plan library it cannot generate one of the pending sets we are interested in and therefore can be filtered from consideration. The execution traces can also be filtered to be consistent with the unobserved actions that are implied by unenabled actions and observed state changes.

To summarize then, we handle hostile agents by extending the observed sequence of actions with hypothesized unobserved actions consistent with both the observed actions, observed state changes, and the plan graph to create a set of possible execution traces. Then we follow the plan recognition algorithm as before. We use the set of execution traces to construct the pending sets and then the probability distribution over the sets of hypotheses of goals and plans implicated by each of the traces and pending sets.

### 7.4. Example

The following example will illustrate this algorithm at a high level. Consider the following set of action and state change observations with a bound of three unobserved actions.

(**break-in**,**deleted-logs**)

Given these observations and the bound on unobservable actions, the algorithm (implemented in Poole's PHA [14]) walks forward through the list of observations, adding unobserved actions as required to build a set of consistent execution traces. To explain the given observations requires the introduction of two unobserved actions, one to enable the action **break-in** and one to cause the state change reported in **deleted-logs**. The complete process results in 9 possible execution traces. The first:

(**recon**,**break-in**,**clean**,**deleted-logs**)

is consistent with the agent not having executed any further unobserved actions beyond those required to justify the observations. This trace is only consistent with the high level goals of **theft** or **vandalism**.

There are four traces that are consistent with the execution of a second unobserved **recon** action performed at various points in the sequence.

(**recon**,**recon**,**break-in**,**clean**,**deleted-logs**)
(**recon**,**recon**,**break-in**,**clean**,**deleted-logs**)

(**recon**,**break-in**,**recon**,**clean**,**deleted-logs**)
(**recon**,**break-in**,**clean**,**deleted-logs**,**recon**)

These traces are all consistent with the goal of pursuing any two of the top level goals concurrently. Note that in all of these traces the two instances of **recon**contribute to different goals. This explains first two traces that appear to be the same. They result from different execution orderings of the high level goals.

Of the four remaining traces:

(**recon**,**break-in**,**gain-root**, **clean**,**deleted-logs**)
and
(**recon**,**break-in**,**clean**,**deleted-logs**,**gain-root**)

are consistent only with the goal of **theft**. Note the ordering differences due to the partial ordering in the plan library. The final two execution traces:

(**recon**,**break-in**,**mod-webpage**,**clean**,**deleted-logs**)
and
(**recon**,**break-in**,**clean**,**deleted-logs**,**mod-webpage**)

are consistent only with the goal of **vandalism**. Again, note the ordering differences due to the partial ordering in the plan library.

In constructing this set of possible execution traces PHA has already established a probability distribution over the explanations and establishes the most likely goal. In this case, since the number of explanations for **theft** and **vandalism** are equal and there are no environmental factors that would weigh in favor of one over the other, these goals are equally likely. The conjunctive plans of **theft** or **vandalism** with **info** is a much less likely third alternative.

### 7.5. Assumptions

In our implementation of this algorithm we have made two assumptions about the observation stream:

1. There is a fixed and known upper bound on the number of unobserved actions.
2. The given observations are true and correctly ordered.

Neither of these assumptions is strictly necessary. We will consider each of them in turn.

Bounding the number of possible unobserved actions enables reasoning about where the agent could be in the execution of its plans. Suppose we bound the number of unobserved actions at two, and we observe a **break-in** action. This observation is not consistent with the agent having already executed **steal**. We have seen one action and the agent may have executed two more unobserved. The agent can have executed a total of three actions. Since, **steal** is the fourth step in its plan, the agent could not yet have executed it.

This bound can be removed from the algorithm in a number of ways including: running the algorithm multiple times with increasing bounds or replacing the bound with a probability distribution over the number of unobserved actions and weighing the execution traces accordingly. We see determining the best way to remove this limitation as an area for future work.

Second, we assumed that the observed actions happen and in the order indicated by the sequence. Thus if we have a sequence of three observations: **recon**, **break-in**, and **gain-root**, we know **recon** happened before **break-in** which happened before **gain-root**. The observation sequences are **not** assumed to be complete, therefore we can't conclude **clean** *didn't* happen between **break-in** and **gain-root** or even after **gain-root**. Ordering constraints provided by the plan library do allow us to rule out some possibilities. For example, the ordering constraints allow us conclude that if **clean** did occur unobserved it couldn't have occurred before the **break-in** unless there were an earlier *unobserved* **break-in**.

This assumption means we need not question the validity of observations. In environments with hostile agents, this assumption must be questioned. Consider a military example, if we receive a report of troops massing at a particular location, we must first determine the validity of the report before considering the effect this would have on our assessment of the enemy's goals. It is straightforward to complicate the model by including a traditional model of noisy observations.[9]

## 8. Conclusions

In this paper we have argued for extending IDSs with a probabilistic model of plan recognition. We have identified many requirements that are placed on the process of plan recognition by the network security domain and have shown how our model of plan recognition based on plan execution meets these requirements. These extensions remove a major assumption of previous research in plan recognition and significantly broadens the domains where plan recognition can be applied.

In future work we are interested in more advanced forms of misdirection. Our current model still does not provide a facility to represent that an agent may be engaging in actions solely to mislead the observer. Our model currently must conclude that the agent does in fact intend to perform the actions for the specified goal. In a sense this is correct; the agent does intend the actions but not the resulting goal. The agent only intends to mislead the observer rather than achieve the goal.

Finally, and perhaps most importantly for our domain, we are interested in disambiguating the goals of multiple agents within the same observation stream. Any computer

network administrator would be thrilled if they only had to deal with a single hacker at a time, however this is simply not the case. For our work to be truly useful in this domain we must be able to disambiguate multiple agents that may be working together or separately.

## Acknowledgments

Support for this work was provided by DARPA/AFRL through contract number F30602-99-C-0177. Thanks to David Poole for providing his Probabilistic Horn Abduction PHA interpreter and help and guidance on working with it.

## References

[1] "Intrusion Detection Message Exchange Format," `http://search.ietf.org/-internet-drafts/-draft-ietf-idwg-idmef-xml-01.txt`.

[2] E. Charniak and R. P. Goldman, "A Bayesian Model of Plan recognition," *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, November 1993.

[3] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison Wesley, Reading, MA, 1985.

[4] P. R. Cohen, C. R. Perrault, and J. F. Allen, "Beyond Question Answering," in *Strategies for Natural Language Processing*, W. Lehnert and M. Ringle, editors, pp. 245–274, Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

[5] K. Erol, J. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning," in *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*, K. J. Hammond, editor, pp. 249–254, Los Altos, CA, 1994, Morgan Kaufmann Publishers, Inc.

[6] R. Feiertag, C. Kahn, P. Porras, S. Schnackenberg, S. Staniford, and B. Tung, "A Common Intrusion Detection Language (CISL)," Available at: `http://www.gidos.org/drafts/language.txt`.

[7] M. Georgeff and A. Lansky, "Procedural Knowledge," *IEEE Special Issue on Knowledge Representation*, vol. 74, pp. 1383–1398, October 1986.

[8] R. P. Goldman, C. W. Geib, and C. A. Miller, "A New Model of Plan Recognition," in *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, 1999.

[9] R. P. Goldman, W. Heimerdinger, S. Harp, C. W. Geib, V. Thomas, and R. Carter, "Information Modeling for Intrusion Report Aggregation," in *Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX-II 2001)*, 2001.

[10] M.-Y. Huang and T. M. Wicks, "A Large-sace Distributed Intrusion Detection Framework Based on Attack Strategy Analysis," in *Recent Advances in Intrusion Detection (RAID98)*, 1998.

[11] M. J. Huber, E. H. Durfee, and M. P. Wellman, "The Automated Mapping of Plans for Plan recognition," in *Uncertainty in Artificial Intelligence,Proceedings of the Tenth Conference*, R. L. de Mantaras and D. Poole, editors, pp. 344–351. Morgan Kaufmann, July 1994.

[12] F. Ingrand, M. Georgeff, and A. Rao, "An Architecture for Real-Time Reasoning and System Control," *IEEE Expert*, vol. 7:6, pp. 34–44, dec 1992.

[13] H. Kautz and J. F. Allen, "Generalized plan recognition," in *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 32–38, 1986.

[14] D. Poole, "Logic Programming, Abduction and Probability: a top-down anytime algorithm for stimating prior and posterior probabilities," *New Generation Computing*, vol. 11, no. 3–4, pp. 377–400, 1993.

[15] D. V. Pynadath and M. P. Wellman, "Generalized Queries on Probabilistic Context-Free Grammars," To appear in IEEE *PAMI*, 1997.

[16] C. Schmidt, N. Sridharan, and J. Goodson, "The plan recognition problem: an intersection of psychology and artificial intelligence," *Artificial Intelligence*, vol. 11, pp. 45–83, 1978.

[17] C. L. Sidner, "Plan parsing for intended response recognition in discourse," *Computational Intelligence*, vol. 1, no. 1, pp. 1–10, 1985.

[18] M. Vilain, "Getting Serious about Parsing Plans: A Grammatical Analysis of Plan Recognition," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 190–197, Cambridge, MA, 1990, MIT Press.

[19] M. P. Wellman and D. V. Pynadath, "Plan Recognition under Uncertainty," Unpublished web page, 1997.

[20] R. Wilensky, *Planning and Understanding*, Addison-Wesley, Reading, MA, 1983.