Real-Time Goal Recognition Techniques for Intrusion Detection using Attack Graphs

Reuth Mirsky and Ya'ar Shalom and Ahmad Majadly and Ya'akov (Kobi) Gal and Rami Puzis
Ben-Gurion University of the Negev, Israel
{dekelr,yaarsh,ahmadmaj,kobig,puzis}@bgu.ac.il

Abstract

Attack graphs are a common domain representation used as the basis for static analysis of potential attacks and defenses in cyber security. So far they have not been used for online goal recognition. This paper bridges this gap by using attack graphs as a basis for goal recognition algorithms to potentially improve intrusion detection and incident response. It provides novel methods to deal with noise and partial observability that are common properties of intrusion detection scenarios in the real world. It compares the efficacy of several goal recognition paradigms, including those using planning algorithms to guide the search, as well as distance-based metrics using attack graphs as an underlying domain representation. Results on a real network of a large scale academic organization show that there is a clear tradeoff between the runtime of the algorithm and its ability to recognize attacks given noisy and partial information. However, using our proposed techniques, most of the computation efforts are invested in preprocessing, thus enabling the online component to perform well in real-time.

Introduction

As computer networks increase in size and complexity, so does the complexity of attacks on these networks. A challenging task that emanates from the world of cyber security is intrusion detection. An Intrusion Detection System (IDS) is meant to monitor and detect malicious activity in a network. A common representation of the protected network used by some IDSs is *Attack Graphs* [Noel and Jajodia, 2004; Shmaryahu *et al.*, 2017]. An attack graph is a description of a network that comprises hosts, their vulnerabilities, and their connectivity to one another. This representation has been used for off-line optimization of a network and to detect vulnerabilities that might lead to an intrusion, or to correlate between different alerts that arise in the network in real-time [Wang *et al.*, 2006].

When an agent attacks the system, we would like to detect the intrusion as fast as possible. Moreover, we would like to be able to reason about the **goal** of the attacker in order to counter the attack while it is still possible. Mapping a network to an attack graph has not been used for real-time attack goal recognition.

Goal recognition is a key AI problem that deals with reasoning about an agent's goals according to a sequence of observed actions. A goal recognition algorithm allows an *observer* (the security operations center) to reason about the goals and execution process of the *actor* (the attacker) given a domain representation (the attack graph) and a set of observed actions (the IDS alerts).

Such an algorithm receives a sequence of observations and a domain description as input and outputs either a goal or a distribution over all possible goals [Bui, 2003; Blaylock and Allen, 2006; Bisson *et al.*, 2011]. While several works have tried to reason about probabilistic goal recognition for cyber security, they were evaluated theoretically, under various relaxations that are not realistic in the real-world, or using toy problems [Geib and Goldman, 2001; Bisson *et al.*, 2011; Mirsky *et al.*, 2017a; Goldman *et al.*, 2018]. Moreover, IDSs usually produce non-negligible number of false positive and false negative alerts, so the goal recognition algorithm used must be robust to faulty observations.

In this paper, we leverage the representational power of attack graphs in order to perform real-time attack goal recognition in a network of a large scale academic organization.

State of the art planning techniques for goal recognition [Ramırez and Geffner, 2009; Sohrabi *et al.*, 2016; Pereira *et al.*, 2017b; Vered *et al.*, 2018] require running a planner as part of the recognition task. Running a planner is a computationally intensive task, making stat of the art techniques impractical for the task of real-time goal recognition. We hence propose a few techniques for faster goal recognition that only require using planners for preprocessing but not for the online queries preformed in real-time. This work is the first to use attack graphs for online goal recognition.

Figure 1 shows the proposed approach, working in a pipeline with existing intrusion detection approaches that correlate and aggregate alerts [Chyssler *et al.*, 2004; Al-Mamory and Zhang, 2007]. The bottom of the figure shows the target computer network.A PDDL representation of the attack graph (Figure 1 left) is generated from the network using standard tools. Alerts generated by IDS deployed within the network are aggregated and compiled into observations (Figure 1 middle). Both, the alerts and the attack graph are the inputs for the goal recognition algorithm (Figure 1 top). The goal recognition algorithm outputs a distribution over the goals (Figure 1 right), which can later be used to refine

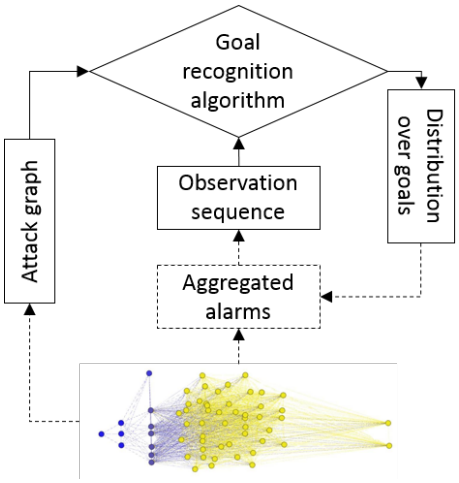

Figure 1: Components of the Proposed Approach.

the alert aggregation and correlation performed by the IDS. The components presented in this work are represented with solid lines, where other component which are not in the focus here are represented using dashed lines.

By providing a synergy between attack graphs and goal recognition, this work presents the following contributions:

1. Adapting goal recognition for attack graphs.
2. A set of algorithms for goal recognition that can handle partial observability and give real-time performance.
3. An evaluation on a real-world network, including partial observability and noise.

Related Work

Attack Graphs

Swiler *et al.* [2001] have presented an automatic tool for generating an attack graph representation of a computer network. This work has been extended to handle networks at larger scale [Noel and Jajodia, 2004]. Later Noel and Jajodia [2008] presented a method for optimizing the placement of intrusion detection system (IDS) sensors and prioritizing IDS alerts using attack graph analysis.

Recent studies continued improving the attack graph generation and analysis approaches toward automated pentesting [Hoffmann, 2015; Durkota *et al.*, 2015; Gonda *et al.*, 2017; Shmaryahu *et al.*, 2018]. Hoffmann [2015], discusses the the suitability of the "CyberSecurity" benchmark at the International Planning Competition (IPC) and analyzes the importance of factoring uncertainty when it comes to understanding the behavior of potential hacking agents. However, analysis of an attack graph has only focused on mapping of vulnerabilities and pentesting, rather than on real time intrusion detection.

A different line of research does utilize the attack graphs for prioritizing alerts generated by IDS. These works use attack graph for alert correlations [Noel *et al.*, 2004; Wang *et al.*, 2006; Zhang *et al.*, 2008; Roschke *et al.*, 2011]. Modern attacks are getting more complex and the number of alerts emerging from the system increases significantly. Reasoning about temporal order and causality of alerts, allows detecting false negatives and false positives more efficiently Noel *et al.* [2004]; Roschke *et al.* [2011]. This line of research has done a great deal in refining the alerts, but did not reason about the ultimate goal of the attacker.

In current work the attack graph is used for online goal recognition. In order to do so, we feed the alerts as observations into a goal recognizer with the attack graph as the underlying domain description.

Goal Recognition

There are several approaches to represent a domain in goal recognition, including policy-based, library-based and others [Avrahami-Zilberbrand and Kaminka, 2005; Vered *et al.*, 2018]. For example, YAPPR [Geib *et al.*, 2008] takes as input a plan library and can output both a distribution over the goals of the actor and predictions about future actions. DOPLAR [Kabanza *et al.*, 2013] extended YAPPR using probabilistic reasoning to reach better performance, at the cost of completeness. These works require to model the settings using a plan library, which is difficult to elicit and susceptible to faults. Bui [2003] uses particle filtering to provide approximate solutions to goal recognition problems. These works all rely on a model of the plans or strategies that the agent can execute.

Some recent advent of work on goal recognition as planning takes as input a planning domain, usually described in PDDL, a set of possible goals. Its output is one of the goals or a distribution over all possible goals [Ramırez and Geffner, 2010; Sohrabi *et al.*, 2016; Freedman and Zilberstein, 2017; Shvo *et al.*, 2017; Vered and Kaminka, 2017; Masters and Sardina, 2017]. The benefit of this approach is that it is model-free, in the sense that possible plan executions are implicit. This compact representation allows an acting agent great flexibility, but makes the recognition of the goals harder. Another recent heuristic goal recognition works include landmark-based heuristics for goal recognition [Pereira *et al.*, 2017a] and cost propogation in order to estimate goal probabilities [Martin *et al.*, 2015].

While adversarial goal recognition was investigated in the past (e.g., [Geib and Goldman, 2001; Lisý *et al.*, 2012; Le Guillarme *et al.*, 2016; Mirsky *et al.*, 2017a]), none of the works mentioned above have combined the PDDL description that can represent an attack graph in order to recognize the goals of an attacker.

Background

We start by briefly mentioning some basic concepts in the world of goal recognition, simplified for brevity. For a more detailed and formal description, we refer the reader to [Ramırez and Geffner, 2009].

In a goal recognition problem, there is an observer and an actor, and the observer needs to infer the goal of the actor. The observer is given a domain description, describing the possible actions that the actor can execute.

Definition 1 *A **Domain Description** (D) is a tuple* $L = \langle S, A, G, I \rangle$*, where* S *is a set of states,* A *is a set of actions*

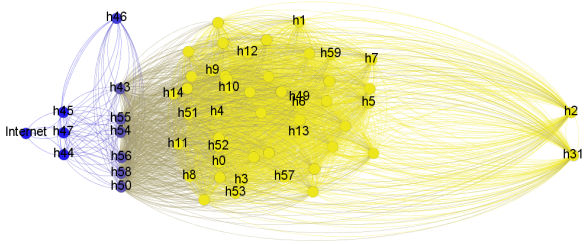

Figure 2: An illustration of the Network Used in This Work

representing transitions from state to state, $G \subseteq S$ is the goals the actor can achieve and $I \in S$ is the initial state of the settings.

Each state in S is represented by a set of predicates. All predicates in the state are assumed to hold true in the environment, and predicates that do not appear in the state are assumed to be false. Each action a in A has preconditions and effects, describing what should be true in the state before executing a and after its execution, respectively. They are both represented as a set of predicates.

Definition 2 *A **plan** for achieving a goal $g \in G$ is a sequence of actions $a_1, \ldots, a_n$ such that:*

- *The execution starts at I (the predicates in the precondition of the first action are all in I).*
- *For each $a_i \in A$, the state before the execution contains all of the predicates in a_i's precondition.*
- *After executing all actions in the sequence, all the predicates in g are true.*

Similarly, an **observation sequence** is a partial plan. The actor is assumed to plan by choosing a goal and then carrying out a plan for reaching this goal, given a set of actions.

In this work, we assume that the agent is rational and tries to achieve exactly one goal at a time. However, we allow for the observation sequences to be noisy (to model possible false positives) and to have missing actions (to model possible false negatives).

Another relaxation made in this work is that all goal recognition techniques assume a single optimal plan. There have been works on goal recognition as planning that reason about k-best plans instead of a single one [Sohrabi *et al.*, 2016]. However, running a planner k times becomes costly in terms of time, which is less desireable for real-time intrusion detection. While this might seem like a serious compromise, it is worth mentioning that in many cases, the difference between plans achieving the same goal is much smaller than plans for achieving a different goal (as shown in Mirsky *et al.* [2017b]), which means that even if the actor is executing a different optimal plan than the one we chose, it is more similar to the plan chosen for the real goal than to plans of other goals. This makes the use of a single optimal plan a reasonable standard for evaluation.

We will use a running example based on our intrusion detection domain. In this example, we have a network of computers as shown in Figure 2. Each node represents a host on the network, and vertices represent the connectivity of the hosts. The leftmost node is the internet, and the three nodes connected to it (the second layer from the left) are three possible access points to the network. The two rightmost nodes are possible hosts an attacker might wish to reach.

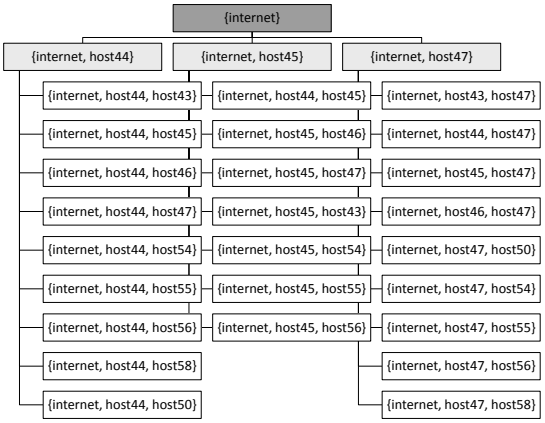

Figure 3: A Decomposition of the First Three Possible States

Figure 3 shows a decomposition of the first three possible states of the world (invariants are omitted for brevity). Each state is represented by the list of computers that are accessible to the acting agent. The top, darkest node is the initial state, where the acting agent only has access to the internet. The next layer, with three nodes, represent the next three states that the agent can reach, depending on the action executed.

Given that $host44$, $host45$ and $host47$ (respectively labeled $h44$, $h45$, $h47$ in Figure 2) are the access points to the network, the agent will need to gain access to one of these hosts. Assuming that we have an action in the domain description called $GetAccess(source, target)$, if the agent executes $GetAccess(internet, host44)$ the next state of the system will be that we have reached the leftmost node in the second layer, where the agent has access to $\{internet, host44\}$. The above action is only executable if the following preconditions hold:

1. The agent has access to the internet (in our domain, as described in the domain section, this is represented as a predicate $(accessible\ \ internet)$).
2. The nodes $internet$ and $host44$ have a connection between them (in our domain, this is represented by the predicate $(hacl\ \ internet\ \ host44)$.

The effect of executing this action is that now the agent has access to host44.

Goal Recognition Techniques

All of the algorithms and techniques used in this work rely on the same input and output: the input is a domain description, represented in PDDL format and an observation sequence. The output is a distribution over the goals.

Formally, given a domain with n possible goals, $g_1, \ldots, g_n$ and an observation sequence $O = o_1, \ldots, o_t$, a goal recognition algorithm estimates the probabilities that the agent is trying to pursue each of the goals, $p(g_i \mid O)$.

We now detail the different algorithms used in this work, given the same input and output as described above.

Original Goal Recognition as Planning

The first algorithm we evaluate is the algorithm by Ramırez and Geffner [2009], which we will refer to as R&G. This algorithm provides an estimation for a goal's probability, by computing a heuristic of likelihood. Intuitively, $L(g_i \mid O)$ measures how close (in terms of cost) is the plan the agent executed from the optimal plan for g_i. This estimation is then normalized over all goals to get a valid distribution p.

In order to estimate $L(g_i \mid O)$, R&G uses an off-the-shelf classical planner in order to calculate two quantities:

1. The cost (number of steps to be taken) of achieving the goal g_i while passing through all the observations. This cost is denoted $C_i(O)$.
2. The cost of achieving the goal g_i without passing through all the observations (note that passing through all but one is valid). This cost is denoted $C_i(\neg O)$.

With these two quantities, the likelihood depends on the difference between a plan that passed through the agents observed actions and the plan that does not go through these observations to achieve the goal g_i:

$$L(g_i \mid O) = e^{\beta(C_i(O) - C_i(\neg O))} \tag{1}$$

Where β is used to soften the impact of partially observing non-optimal plans. This heuristic is then transformed to the probability p_i, where we follow the representation from Geffner and Bonet [2013], assuming we have an a priori uniform distribution over the goals:

$$p(g_i|O) = \frac{1}{L(g_i \mid O) + 1} \tag{2}$$

After $p(g_i|O)$ is calculated for all $g_i \in G$, they are normalized to provide a valid probability distribution.

Sunk Cost Variation

As shown in the empirical section, the R&G algorithm is not susceptible to noise. We remind that this work is aimed to provide goal recognition of an intrusion detection. As such, the attacker might wish to obfuscate the attack by executing irrelevant actions, or the alert aggregation might produce false positive alerts which are not part of a valid attack.

Consider the following two scenarios: in the first scenario, a potential attacker has executed nine out of ten steps of an attack $o_1, \ldots, o_9$, and then did a tenth, not related action, $\bar{o}$. In the second scenario, a potential attacker has executed the first out of the ten steps of the attack, o_1, and then executed the second, not related action $\bar{o}$.

According to the R&G algorithm with $\beta = 0$, when we try to calculate the probability that the agent is trying to complete the same attack (we'll call it g), we get that:

$$L(g \mid o_1, \ldots, o_9, \bar{o}) = L(g \mid o_1, \bar{o}) = e^{11-10} = 2.718$$

and respectively

$$p(g \mid o_1, \ldots, o_9, \bar{o}) = p(g \mid o_1, \bar{o}) = 0.27$$

This happens because the cost of executing the task is the same, no matter when the interruption (the noisy action) happens. However, in practice, we understand that if a potential attacker has already put the effort and executed nine out of ten steps, it is more likely that this is an actual attack than the case when only one action is relevant.

To reason about the problem described about, we propose a different, non-constant value for the β parameter in the likelihood heuristic that takes into account the cost the attacker had paid for executing the possible attack so far (which we refer to as sunk cost):

$$\beta = \frac{1}{min(C_i(\neg O), C_i(O))}. \tag{3}$$

Thus, we get the following likelihood heuristic:

$$L_{sunk}(g_i \mid O) = e^{\frac{C_i(O) - C_i(\neg O)}{min(C_i(\neg O), C_i(O))}} \tag{4}$$

Given this modification, we get that now the likelihood of the two observation sequences is different:

$$L_{sunk}(g \mid o_1, \ldots, o_9, \bar{o}) = e^{\frac{11-10}{10}} = 1.105$$
$$p_{sunk}(g \mid o_1, \ldots, o_9, \bar{o}) = 0.48$$

while

$$L_{sunk}(g \mid o_1, \bar{o}) = e^{\frac{11-10}{2}} = 1.65$$
$$p_{sunk}(g \mid o_1, \bar{o}) = 0.38$$

.

Plan Edit Distance

One more special trait of goal recognition for intrusion detection is that we wish to be able to respond in **real-time** to potential attacks. This means that the runtime of the goal recognizer is key for good performance. Both of the methods described above require to run a planner at least once online, after we observe the actions: if O is not part of the optimal plan, we will need to run a planner to calculate $C_i(O)$ and if all observations O are part of the optimal plan, we will need to run the planner to get $C_i(\neg O)$. Since running a planner can be costly, we propose another measure to evaluate the cost of reaching each of the goals, given an observation sequence, that do not require online planner executions.

Using plan edit distance, we give a plan a cost that uses a naive heuristic regarding the distance of a plan from each of the goals' optimal plans. Given an observation sequence O and a goal g we define $D(g, O)$ to be the edit distance of the actions between the optimal plan and the prefix we observe. Note that the edit distance can be calculated in several ways, while assigning different costs to addition of actions from removal of actions, or for accounting about the order of the actions. We show here only the most basic calculation where

the distance between $A = a_1, \ldots, a_n$ and $A' = a'_1, \ldots, a'_k$ is:

$$D(A, A') = |(A \cup A') \setminus (A \cap A')| \quad (5)$$

This metric reasons about the number of actions already executed, preferring plans which have executed more steps from the optimal plan. It also does not require to run a planner online: the planner is only needed to find the optimal plans, a task that can be done offline. On the other hand, this metric is very naive, as it does not reason about the order by which the actions have taken place, or the actual cost of executing another action.

Given this distance metric, we calculate the probability of an observation sequence O given g_i (and an optimal plan for this goal O^*) as follows:

$$p(g_i|O) = \frac{1}{D(O, O^*) + 1} \quad (6)$$

Like in the previous algorithms, after $p(g_i|O)$ is calculated for all $g_i \in G$, they are normalized.

Alternative Plan Cost Variation

The last technique is based on the work of Felner *et al.* [2007] to calculate the costs of alternative plans. Given a distance metric between states, it computes the similarity of a prefix to a complete sequence. Unlike the plan edit distance method, this technique can capture more complex relations between actions and states, assuming it is given a good distance metric.

Landmarks are facts that must be true at some point in every valid solution plan [Hoffmann *et al.*, 2004]. They are used to represent in an informed way how many of the desired predicates from the goal state we have reached, and are commonly used in planning.

We use landmarks to provide the following distance metric between states in a planning domain:

$$D_g(s_1, s_2) = |(L_g(s_1) \cup L_g(s_2)) \setminus (L_g(s_1) \cap L_g(s_2))| \quad (7)$$

Where s_1, s_2 are states, g is a goal and $L_g(s)$ is the list of landmarks for g that were achieved in s.

This metric is different for each goal, but once the landmarks are extracted (a process that can be done offline), in real-time the only calculation required is counting the number of landmarks per state reached using the executed plan (which is done by sequentially by collecting the effects of the actions).

Next, we can create a mapping between the states of the two plans (the optimal and the observed). Using the distance metric described about, we can compute the cost of the mapping by summing the costs of the mapped states according to D_g.

Out of the several mappings in the original paper by Felner *et al.* [2007], we chose the Time Dimension Mapping. This mapping is done according to the time dimension, i.e., each state in the original sequence of states is mapped to its relative state in the destination sequence according to the proportion of its relative location on the path. This gives us the following equation:

$$C_s(O|g) = \operatorname*{argmin}_M \frac{\sum_{i=1}^{k} D_g(M(s_i), s_i)}{k} \quad (8)$$

Where M is a conjunctive mapping from $s_1, \ldots, s_k$ to $s^*_1, \ldots, s^*_n$. Intuitively, we choose the mapping with the smallest sum of distances (in terms of achieved landmarks) between the traversed states of the observation sequence and their mapped states in the optimal plan.

An important property of this mapping is that it is monotonic, meaning that given a mapping M with $M(s_i) = s^*_j, M(s_{i+l}) = s^*_k$, it must hold that $j < k$. This property makes sure that we reason about the order in which the actions are executed in the observation sequence to match the order of the optimal plan.

However, summing the cost of this mapping might miss important information if the observation sequence is shorter than the optimal plan ($k < n$). In order to cover the remaining actions in the optimal plan ($k + 1, \ldots, n$) as well, we provide another cost function which performs the opposite mapping from $s^*_1, \ldots, s^*_n$ to $s_1, \ldots, s_k$ in a similar way. This function is denoted $C_{s^*}(O|g)$.

In order to reason about both scenarios, that either the observation sequence or the optimal plan is longer, we calculate the averae cost of both functions. Thus, we get the following cost function:

$$C(O|g) = \frac{C_s(O|g) + C_{s^*}(O|g)}{2} \quad (9)$$

This cost function reasons about the length of the observation sequence by summing the cost of two conjunctive mappings that while each of the two mappings maintains the monotonicity property.

Finally, the probability of an observation sequence O given a goal g_i is calculated as

$$p(g_i|O) = \frac{1}{C(O|g) + 1} \quad (10)$$

Like in previous techniques, the probabilities are normalized to provide a valid distribution.

Intrusion Detection Domain

The domain is based on the work of Shmaryahu [2016], and was compiled to match the requirements of our goal recognition techniques. The domain has 4 types of variables, $host, os, sw, vuln$, representing hosts, operating systems, software and vulnerabilities respectively.

These types are used to define 7 parameterized predicates:

1. $(hacl \;\; ?src - host \;\; ?target - host)$ - true if src and $target$ are linked.
2. $(controlling \;\; ?h - host)$ - true if the agent controls h.
3. $(HostSW \;\; ?src - host \;\; ?s - sw)$ - true if host src runs software s.
4. $(HostOS \;\; ?src - host \;\; ?o - os)$ - true if host src runs operating system o.

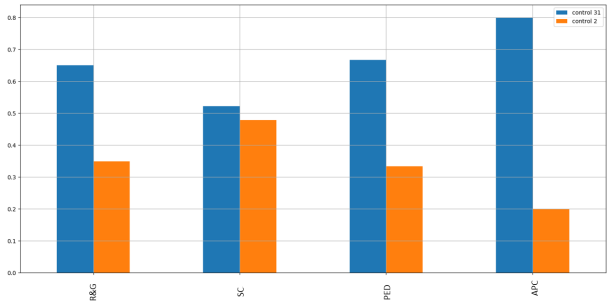

Figure 4: Probability Distribution for Multiple Goals

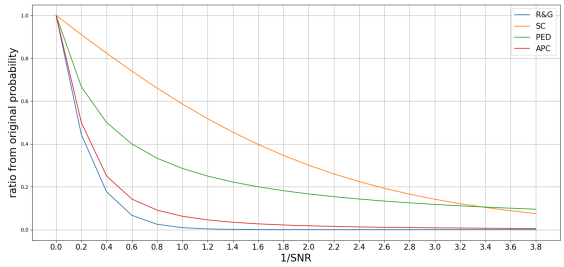

Figure 5: Probability Decline with Noisy Observations (False Positives)

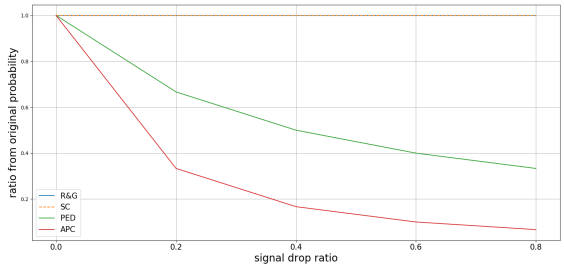

Figure 6: Probability Decline with Missing Observations (False Negatives)

5. $(ExistVuln\ ?v-vuln\ ?target-host)$ - true if vulnerability v exist in host $target$.
6. $(Match\ ?o-os\ ?s-sw\ ?v-vuln)$ - true if vulnerability v exist in software s under operating system o.
7. $(accessible\ ?target-host)$ - true if host $target$ is accessible by the agent.

In addition, the domain defines the actions the agent can perform:

- $GetAccess(src, target)$, which has the precondition that the source and target hosts are linked, and the effect is that host $target$ is now accessible to the agent.
- $RunSW(target, sw, os, v)$ is the action used to run a software sw on the system os of host $target$, that might exploit a vulnerability v. This combination might be a valid software usage, or an exploit.

Using these actions, reaching a goal is a process of gaining access to a host, then running some software on it. This is done repeatedly, until reaching the host we wish to control.

The specific attack graph we use in our evaluations contains 60 hosts. The network architecture, operating systems and softwares ran by each host, and the goal host are defined by the original network scans, but the vulnerabilities data was randomly generated so that each host has 15 possible vulnerabilities.

As collecting real attacks on the network is not feasible, we simulated attacks on the attack graph by using the HSP planner [Bonet and Geffner, 2001]. We then injected noisy observations to the plans to simulate false positive alerts and removed observations to simulate false negative alerts.

Empirical Evaluation

This section detail the results of running the 4 goal recognition techniques on a home-commodity $i7$ computer. All algorithms are implemented in Python and the domain is encoded in PDDL. The algorithms are the original goal recognition as planning (denoted R&G), sunk cost (denoted SC), plan edit distance (denoted PED) and alternative plan cost (denoted APC).

First, we examined whether the probability can be a true representative of the actual goal recognizer. Figure 4 shows the probability distribution when there are 2 possible goals to be recognized. Goal number 1 (blue) is the real goal behind the observation sequence. As seen in this figure, all of the algorithms gave it the highest probability, meaning that they managed to capture the true goal.

Next, we looked at different false information that can interrupt the recognition process. When trying to perform an intrusion, an attacker is likely to try and obfuscate the attack. These efforts can lead to missing observations (False Negative alerts) or noisy observations (False Positive alerts).

The first property examined is the algorithms' sensitivity to noisy observations. Noise can be a result of faulty sensors, false positive alerts or when multiple plans are executed in the same time.

Figure 6 shows the probability decline for all recognition techniques as more observations are noisy. The x-axis shows the Signal to Noise Ration (SNR) which represents the number of observations that were injected to the original plan in comparison to the length of the original plan, and the y-axis represented the percent from the probability of that original plan that the noisy plan received.

As seen from this figure, R&G is most susceptible to noise and the probability declines fast when introducing noisy observations. SC is the best to deal with noise until a very late stage - it declines as well, but slower due to the plan-length normalization of the cost. When reaching a very high SNR (where there are 3.4 false positive per 1 true positive), the PED technique prevails. We believe that this is due to the fact that there is a finite number of landmarks that can be achieved, and after a certain point this list does not change much, and the best mapping gives a constant cost to every new noisy observation.

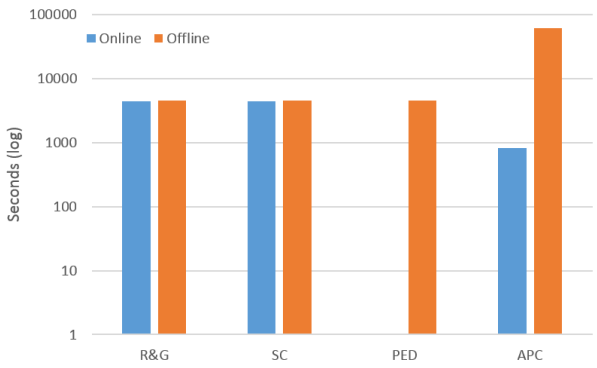

Figure 7: Online and Offline Runtime on Complete Fully Observable Plan

The second property examined is the algorithms' sensitivity to missing observations. Such observations can be the results of the attacker succeeding in hiding their actions from the alert system, or the fact that the attack is still ongoing and not all of the attack steps has been executed yet.

Figure 6 shows the probability decline for all recognition techniques as more observations are missing. The x-axis is Signal Drop Ratio (SDR), representing relative proportion of the number of observations that were removed from the original plan, and the y-axis represented the percent from the probability of that original plan that the partial plan received.

As seen in this figure, goal recognition as planning algorithms are not sensitive to false negatives, as they complete the needed missing observations anyway as part of their solution. However, the plan edit distance and alternative plan cost variations both reason about the proportion of the plan that was executed. Due to this property, there is a decline in the probability using these techniques, where in the alternative plan cost the decline is faster.

Lastly, one of the most interesting desired properties from a goal recognizer in an intrusion detection system, is the ability to make a recognition in real-time. Figure 7 shows the online and offline runtimes of each algorithm. The x-axis is the different algorithms and the y-axis represents the time in log scale of seconds. The real-time computation of each algorithm is different: R&G and SC require to run the planner once more per goal, to compute an alternative plan given the observations. The PED variation is the lightest and requires to compute the difference between the actions in the optimal plan and the observation sequence. The APC variation requires to find the landmarks achieved in the observation sequence (a traversal over the list of landmarks) and then to compute the edit distance of two lists of predicates. These differences in computation affect the runtime as can be seen in Figure 7. The PED is the fastest and APC is about an order of magnitude faster than R&G and SC, as it does not require to run a planner for the second time.

Each algorithm has also a different cost for its preprocessing: All algorithms require to find an optimal plan for each goal. Additional to this, the alternative plan cost variation requires to extract the set of landmarks for each goal, a process that takes more than an order of magnitude longer than the other algorithms, but is performed offline.

Discussion

This paper presented the use of goal recognition techniques on attack graphs for intrusion detection. It utilizes a state-of-the-art goal recognition technique using planning, and then adds three new variations and techniques to tackles specific challenges that are interesting in the context of an intrusion detection system. All of the techniques are evaluated on a real-world network and simulated attacks.

The empirical results show that all techniques manage to deal with false negatives and false positives, while having the highest probability always assigned to the correct goal. However, there is a clear tradeoff between the algorithms in terms of sensitivity to FPs, to FNs and time.

The first two presented techniques (R&G and SC) handle missing observations by extrapolating the observation sequence. This means that they are not susceptible to false negatives, but at the cost of runtime. The PED and APC techniques do not require to run a planner on the observation sequence, thus in real-time the running costs are smaller. The latter does require the extraction of all possible landmarks, but it can be performed offline.

This work opens a series of possible extensions. First and foremost, this work is based on a vision of a complete end-to-end intrusion detection system, comprised from different components. It would be interesting to evaluate the performance of the recognizers when the observations given are actual alerts instead of symbolic, processed tokens.

Acknowledgments

This research was partially funded by the Cyber@Ben-Gurion center. R.M. is a recipient of the Ministry of Science fellowship for Research in Applied Science and Engineering.

References

S.O. Al-Mamory and H. Zhang. A survey on IDS alerts processing techniques. In *The 6th WSEAS international conference on information security and privacy*, 2007.

D. Avrahami-Zilberbrand and G.A. Kaminka. Fast and complete symbolic plan recognition. In *IJCAI*, 2005.

F. Bisson, F. Kabanza, A. R. Benaskeur, and H. Irandoust. Provoking opponents to facilitate the recognition of their intentions. In *AAAI*, 2011.

N. Blaylock and J. Allen. Fast hierarchical goal schema recognition. In *National Conference on Artificial Intelligence*, 2006.

B. Bonet and H. Geffner. Heuristic search planner 2.0. *AI Magazine*, 22(3):77, 2001.

H.H. Bui. A general model for online probabilistic plan recognition. In *IJCAI*, volume 3, pages 1309–1315, 2003.

T. Chyssler, S. Burschka, M. Semling, T. Lingvall, and K. Burbeck. Alarm reduction and correlation in intrusion detection systems. In *DIMVA*, pages 9–24, 2004.

K. Durkota, V. Lisỳ, B. Bosanskỳ, and C. Kiekintveld. Optimal network security hardening using attack graph games. In *IJCAI*, pages 526–532, 2015.

A. Felner, R. Stern, J.S. Rosenschein, and A. Pomeransky. Searching for close alternative plans. *AAMAS*, 2007.

R.G. Freedman and S. Zilberstein. Integration of planning with recognition for responsive interaction using classical planners. In *AAAI*, pages 4581–4588, 2017.

H. Geffner and B. Bonet. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2013.

C.W. Geib and R.P. Goldman. Plan recognition in intrusion detection systems. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 46–55. IEEE, 2001.

C.W. Geib, J. Maraist, and R.P. Goldman. A new probabilistic plan recognition algorithm based on string rewriting. In *ICAPS*, pages 91–98, 2008.

R.P. Goldman, S.E. Friedman, and J.M. Rye. Plan recognition for network analysis: Preliminary report. In *PAIR*. AAAI workshops, 2018.

T. Gonda, G. Shani, R. Puzis, and B. Shapira. Ranking vulnerability fixes using planning graph analysis. In *IWAISe: First International Workshop on Artificial Intelligence in Security*, page 41, 2017.

J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

J. Hoffmann. Simulated penetration testing: From" dijkstra" to" turing test++". In *ICAPS*, pages 364–372, 2015.

F. Kabanza, J. Filion, A. R. Benaskeur, and H. Irandoust. Controlling the hypothesis space in probabilistic plan recognition. In *IJCAI*, pages 2306–2312, 2013.

N. Le Guillarme, A.I. Mouaddib, S. Gatepaille, and A. Bellenger. Adversarial intention recognition as inverse game-theoretic planning for threat assessment. In *ICTAI*, pages 698–705. IEEE, 2016.

V. Lisỳ, R. Píbil, J. Stiborek, B. Bošanskỳ, and M. Pěchouček. Game-theoretic approach to adversarial plan recognition. In *ECAI*, pages 546–551. IOS Press, 2012.

Y.E. Martin, M.D. R-Moreno, D.E. Smith, et al. A fast goal recognition technique based on interaction estimates. In *IJCAI*, 2015.

P. Masters and S. Sardina. Cost-based goal recognition for path-planning. In *AAMAS*, pages 750–758, 2017.

R. Mirsky, Y. Gal, and D. Tolpin. Session analysis using plan recognition. *Workshop on User Interfaces and Scheduling and Planning (UISP)*, 2017.

R. Mirsky, R. Stern, Y. Gal, and M. Kalech. Plan recognition design. In *AAAI*, pages 4971–4972, 2017.

S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Workshop on Visualization and data mining for computer security*, pages 109–118. ACM, 2004.

S. Noel and S. Jajodia. Optimal IDS sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, 2008.

S. Noel, E. Robertson, and S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *Computer Security Applications Conference*, 2004.

R.F. Pereira, N. Oren, and F. Meneguzzi. Landmark-based heuristics for goal recognition. In *AAAI*, 2017.

R.F. Pereira, N. Oren, and F. Meneguzzi. Plan optimality monitoring using landmarks and planning heuristics. In *PAIR workshop in AAAI*, 2017.

M. Ramırez and H. Geffner. Plan recognition as planning. In *AAAI*, 2009.

M. Ramırez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI*, 2010.

S. Roschke, F. Cheng, and C. Meinel. A new alert correlation algorithm based on attack graph. In *Computational intelligence in security for information systems*, pages 58–67. Springer, 2011.

D. Shmaryahu, G. Shani, J. Hoffmann, and M. Steinmetz. Partially observable contingent planning for penetration testing. In *IWAISe: First International Workshop on Artificial Intelligence in Security*, page 33, 2017.

D. Shmaryahu, G. Shani, J. Hoffmann, and M. Steinmetz. Simulated penetration testing as contingent planning. In *ICAPS*, 2018.

D. Shmaryahu. Constructing plan trees for simulated penetration testing. In *ICAPS*, 2016.

M. Shvo, Sohrabi S., and S.A. McIlraith. An ai planning-based approach to the multi-agent plan recognition problem. In *PAIR workshop in AAAI*, 2017.

S. Sohrabi, A.V. Riabov, and O. Udrea. Plan recognition as planning revisited. In *IJCAI*, pages 3258–3264, 2016.

L.P. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *discex*, page 1307. IEEE, 2001.

M. Vered and G.A. Kaminka. Heuristic online goal recognition in continuous domains. In *IJCAI*, pages 4447–4454, 2017.

M. Vered, R.F. Pereira, M.C Magnaguagno, G.A. Kaminka, and F. Meneguzzi. Towards online goal recognition combining goal mirroring and landmarks. AAMAS, 2018.

L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer communications*, 29(15):2917–2933, 2006.

S. Zhang, J. Li, X. Chen, and L. Fan. Building network attack graph for alert causal correlation. *Computers & security*, 27(5-6):188–196, 2008.