Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Hypothesis Exploration for Malware Detection using Planning

Shirin Sohrabi Octavian Udrea Anton V. Riabov
IBM T.J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA
{ssohrab, oudrea, riabov}@us.ibm.com

Abstract

In this paper we apply AI planning to address the hypothesis exploration problem and provide assistance to network administrators in detecting malware based on unreliable observations derived from network traffic. Building on the already established characterization and use of AI planning for similar problems, we propose a formulation of the hypothesis generation problem for malware detection as an AI planning problem with temporally extended goals and actions costs. Furthermore, we propose a notion of hypothesis "plausibility" under unreliable observations, which we model as plan quality. We then show that in the presence of unreliable observations, simply finding one most "plausible" hypothesis, although challenging, is not sufficient for effective malware detection. To that end, we propose a method for applying a state-of-the-art planner within a principled exploration process, to generate multiple distinct high-quality plans. We experimentally evaluate this approach by generating random problems of varying hardness both with respect to the number of observations, as well as the degree of unreliability. Based on these experiments, we argue that our approach presents a significant improvement over prior work that are focused on finding a single optimal plan, and that our hypothesis exploration application can motivate the development of new planners capable of generating the top high-quality plans.

Introduction

Given a set of possibly unreliable observations derived from network traffic and a model that describes the typical lifecycle of malware or other (potentially non-malicious) behaviors of interest, we are interested in applying AI planning to generate a ranked list of hypotheses about the hosts in the network. The generated list can be presented to a network administrator or to an automated system to run further investigation. The most related problem is that of diagnosis of discrete dynamical systems (e.g., (Cassandras and Lafortune 1999; McIlraith 1994; Cordier and Thiébaux 1994; Sampath et al. 1995)).

Recently, several researchers have proposed use of planning technology to address several related class of problems including diagnosis (e.g., (Sohrabi, Baier, and McIlraith 2010; Haslum and Grastien 2011)), plan recognition (Ramírez and Geffner 2009), and finding excuses (Göbelbecker et al. 2010). These problems share a common goal of finding a sequence of actions that can explain the set of observations given the model-based description of the system. However, most of the existing literature make an assumption that the observations are all perfectly reliable and should be explainable by the system description, otherwise no solution exists for the given problem. But that is not true in general (Thorsley, Yoo, and Garcia 2008). For example, even though observations resulting from the analysis of network data can be unreliable, we would still like to explain as many observations as possible with respect to our model.

In 2011, Sohrabi et al. established a relationship between generating explanations, a more general form of diagnosis, and a planning problem with temporally extended goals (Sohrabi, Baier, and McIlraith 2011). In this paper, we build on this to propose a formulation of the hypothesis generation problem for malware detection where observations may be unreliable as an AI planning problem with temporally extended goals. Furthermore, because not all observations can have an equal weight in determining a root cause, we propose a notion of hypothesis "plausibility" under unreliable observations, which we model as plan quality.

While finding the most plausible hypothesis via planning is interesting and by itself challenging, we argue that this is not sufficient given unreliable observations, as shown in our malware detection problem. Hence, we need to develop a method that not only generates the most plausible hypothesis but also generates the top most plausible hypotheses about the hosts in the network. To this end, we propose a method of exploiting LAMA (Richter and Westphal 2010) to generate multiple distinct high-quality plans. We experimentally evaluate this approach by generating random problems of varying hardness both with respect to the number of observations, as well as the degree of unreliability. In the generated problems, we know the ground truth and want to know if the plans that LAMA generates can detect them. Our results show that our approach is viable in malware detection. Furthermore, we hope that the results presented in this paper can motivate the development of new planners capable of generating the top high-quality plans.

Hypothesis Generation for Network Traffic Analysis

Consider the case of enterprise network monitoring. Although such networks are typically equipped with state-of-

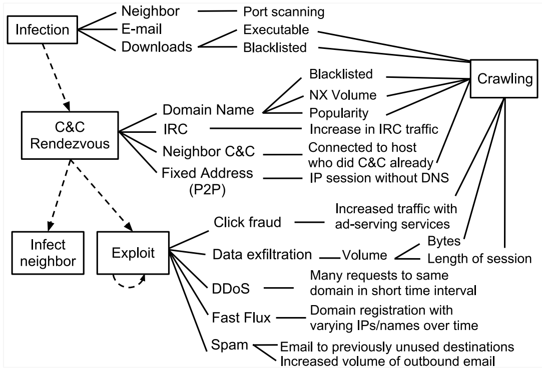

Figure 1: Example of a host's lifecycle.

the-art security appliances and firewalls, these devices and software are designed to look for known signatures of cyber threats by examining very small windows of data, for example, one Transmission Control Protocol (TCP) session. Most modern malware such as *bots* uses techniques for disguising its network communications in normal traffic over time. For effective malware detection, we must employ techniques that examine observations of network traffic over a longer period of time and produce hypotheses on whether such traces correspond to a malware lifecycle or otherwise innocuous behavior. Practically, since any such hypothesis has a degree of uncertainty, it must be presented in a form that explains how observed behavior indicates either malware or other lifecycles of network activity.

As an example, a (simplified) malware lifecycle could be described by a cybersecurity expert as in Figure 1. The rectangles on the left side correspond to malware lifecycle states, such as the host becoming infected with malware, the bot's rendezvous with a Command and Control (C&C) machine (*botmaster*), the spread of infection to neighboring machines and a number of exploits – uses of the bot for malicious activity. Each of the lifecycle states can be achieved in many ways, depending on the type and capabilities of the malware. For example, C&C rendezvous can be achieved by attempting to contact an Internet domain, or via Internet Relay Chat (IRC) on a dedicated channel, or by contacting an infected machine acting a local C&C, or by Peer-to-Peer (P2P) traffic with a preset IP address. In Figure 1, the rightmost nodes (excluding *Crawling*) are observations derived from network data that could be obtained in the corresponding lifecycle states. As an example, *NX volume* corresponds to an abnormally high number of *domain does not exist* responses for Domain Name System (DNS) queries; such an observation may indicate that the bot and its botmaster are using a domain name generation algorithm, and the bot is testing generated domain names trying to find its master. On the right side of Figure 1, we can also see a "normal" lifecycle of a web crawler compressed into a single state. Note that crawler behavior can also generate a subset of the observations that malware may generate. Although this is a small example, it can be extended by adding more known lifecycles, expanding the number of observations covered, introducing loops (e.g., having a periodic C&C rendezvous after exploits), etc. Furthermore, the lifecycle graph can be used to model infection propagation – e.g., by linking the *Infect Neighbor* state for host *A* to *Infection – Neighbor* for host *B* (e.g., if A is port-scanning B for vulnerabilities).

In the lifecycle we described, observations are the result of performing analysis on network data. The *NX volume* observation for instance is the result of modeling the typical number of negative responses to DNS queries, aggregating DNS data for a window of time for a host and determining if its NX response volume is an outlier. While the complexity of the analysis involved to obtain one observation can vary, it is important to note that observations are by nature *unreliable* because of multiple reasons.

The set of observations will be incomplete. Operational constraints will prevent us running in-depth analysis on *all* the network traffic all the time. However, all observations are typically timestamped, and hence totally ordered.

Observations may be ambiguous. This is depicted in Figure 1, where for instance contacting a blacklisted domain may be evidence of malware activity, or maybe a crawler that reaches such a domain during normal navigation.

Observations may be mutually inconsistent. This occurs when we have observations that can be explained by mutually exclusive lifecycle paths – e.g., observations that are exclusive to the malware lifecycle and observations that are exclusive to crawling behavior in the same sequence.

Not all observations will be explainable. There are several reasons while some observations may remain unexplained: (i) in this setting observations are (sometimes weak) indicators of a behavior, rather than authoritative measurements; (ii) the lifecycle description is by necessity incomplete, unless we are able to model the behavior of all software and malware; (iii) there are multiple processes on a machine using the network, making it difficult to determine which process originated which behavior; (iv) malware throws off security scans by either hiding in normal traffic patterns or originating extra traffic to confuse detectors.

Let us consider an example consisting of two observations for a host: (o_1) a download from a blacklisted domain and (o_2) an increase in traffic with ad servers. Note that according to Figure 1, this sequence could be explained by two hypotheses: (a) a crawler or (b) infection by downloading from a blacklisted domain, a C&C rendevouz which we were unable to observe, and an exploit involving click fraud. In such a setting, it is normal to believe (a) is more plausible than (b) since we have no evidence of a C&C rendevouz taking place. However, take the sequence (o_1) followed by (o_3) an increase in IRC traffic followed by (o_2). In this case, it is reasonable to believe that the presence of malware – as indicated by the C&C rendevouz on IRC – is more likely than crawling, since crawlers do not use IRC. The crawling hypothesis cannot be completely discounted since it may well be that a crawler program is running in background, while a human user is using IRC to chat.

Due to the unreliability of observations, in order to address the malware detection problem we need a technique that considers a sequence of observations and produces not just one, but a ranked list of hypotheses (explanations of the observations), such that: (i) some hypotheses may ignore (leave as unexplained) observations that are deemed unreliable or inconsistent and (ii) the rank of the hypothesis is a measure of its plausibility. Since the number of observations in practice can grow into the tens or hundreds for even such small lifecycle descriptions, having a network administrator manually create such a ranked list is not a scalable approach. In this paper, we describe an automated approach that uses planning to automatically sift through hundreds of competing hypotheses and find multiple highly plausible hypotheses. The result of our automated technique can then be presented to a network administrator or to an automated system for further investigation and testing; the testing of hypotheses is outside the scope of this paper.

Modeling

In this section, we first define a dynamical system that can model the host's lifecycle discussed in the previous section. We then define a notion of hypothesis and hypothesis "plausibility". We then establish the correspondence between the hypothesis exploration problem and planning. With this relationship at hand, we turn to a state-of-the-art planner to generate high-quality plans in the following section.

System Description There are many ways to formally define the system description. In this paper, we adopt the definition of a dynamical system by Sohrabi et al. 2011 to account for the unreliable observations. In particular, we model the system by describing the possible transitions in a host's lifecycle similar to the one in Figure 1. To encode these transitions, we use a set of actions with preconditions and effects. To account for the unreliable observations, we introduce an action called "*discard*" that simulate the "explanation" of an unexplained observation. That is, the instances of the *discard* action add transitions to the system that account for leaving an observation unexplained. The added transitions ensures that we took all observations into account but an instance of the *discard* action for a particular observation o indicates that o is not explained.

We define a system to be a tuple $\Sigma = (F, A, I)$, where F is a finite set of fluent symbols, A is a set of actions that includes both actions that account for the transitions in the lifecycle as well as the *discard* action described above, and I is a clause over F that defines the initial states. Note, for the purpose of this paper, we assume the initial state is complete, but our approach can be extended similar to (Sohrabi, Baier, and McIlraith 2010). Actions can be over both malicious and non-malicious behaviors. They are defined by their precondition and effects, over the set of fluents F. A *system state* s (not to be confused with the state in the lifecycle) is a set of fluents with known truth value. For a state s, let $M_s : F \rightarrow \{true, false\}$ be a truth assignment that assigns $true$ to f if $f \in s$ and $false$ otherwise. An action a is *executable* in a state s if all of its preconditions are met by the state or $M_s \models c$ for every $c \in prec(a)$. We define the successor state as $\delta(a, s)$ for the executable actions. The sequence of actions $[a_1, ..., a_n]$ is executable in s if the state $s' = \delta(a_n, \delta(a_{n-1}, \ldots, \delta(a_1, s)))$ is defined; henceforth, is executable in Σ if it is executable from the initial state.

Hypothesis Given the system description $\Sigma = (F, A, I)$ and an observation formula φ, a hypothesis is a sequence of actions $\alpha = [a_1, ..., a_n]$ such that that $a_i \in A$, and the observation formula φ is satisfied by the sequence of actions α in the system Σ. While in general the observation formula φ can be expressed as an Linear Temporal Logic (LTL) formula (Emerson 1990), in this paper we consider the observation formula φ to have the form $\varphi = [o_1, ..., o_n]$, where $o_i \in F$, with the following standard LTL interpretation:[1]

$$o_1 \wedge \bigcirc\Diamond(o_2 \wedge \bigcirc\Diamond(o_3...(o_{n-1} \wedge \bigcirc\Diamond o_n)...))$$

Note that the observations are totally ordered in the above formula. As mentioned earlier, for the problem we are considering, it is typical to have totally ordered observations.

Given a set of observations, there are many possible hypotheses, but some could be stated as more plausible than others. For example, since observations are not reliable, the hypothesis α can explain a subset of observations by including instances of the *discard* action. However, we can indicate that a hypothesis that includes the minimum number of *discard* actions is more plausible. In addition, observations can be ambiguous: they can be explained by instances of "good" (non-malicious) actions as well as "bad" (malicious) actions. Similar to the diagnosis problem, a more plausible hypothesis ideally has the minimum number of "bad" or "faulty" actions. Furthermore, not all observations have an equal weight in determining the root cause. We can have additional knowledge about the domain that could include the likelihood of an event. This additional knowledge can be given as either probabilities or simple ranking over the root causes. For example, observing that a host has downloaded an executable file can raise suspicion of possible infection, but even more so if the host is running Windows.

More formally, given a system Σ and two hypotheses α and α' we assume that we can have a reflexive and transitive plausibility relation $\preceq$, where $\alpha \preceq \alpha'$ indicates that α is at least as plausible as α'. Hypothesis α is the optimal hypothesis for the system Σ and observation formula φ if there does not exist another hypothesis α' such that is more plausible or $\alpha' \preceq \alpha$ and $\alpha \not\preceq \alpha'$.

Relationship to Planning We appeal to the already established relationship between generating explanations and planning (e.g., (Sohrabi, Baier, and McIlraith 2011)). That is given a system $\Sigma = (F, A, I)$ and an observation formula φ expressed in LTL, and the plausibility relation $\preceq$, α is a hypothesis if and only if α is plan for the planning problem $P = (F, A', I, \varphi)$ where φ is a temporally extended goal formula expressed in LTL (i.e., planning goals that are not only over a final state), A' is the set A augmented with positive action costs that captures the plausibility relation $\preceq$. That is if α and α' are two hypotheses, where α is more plausible than α', then $cost(\alpha) < cost(\alpha')$. Therefore, the most plausible hypothesis is the minimum cost plan.

Finding such a cost function in general can be difficult and we do not have a systematic way of finding such cost function. However, some class of plausibility relation can be expressed as Planning Domain Definition Language (PDDL3)

[1] $\bigcirc$ is a symbol for next, $\Diamond$ is a symbol for eventually.

non-temporal preferences (Gerevini et al. 2009) and compiled away to action costs using the techniques proposed by Keyder and Geffner (Keyder and Geffner 2009).

Computation

In the previous section, we established the relationship between planning with temporally extended goals, and the generation of hypotheses for the malware detection problem. That is, we showed that the generation of a hypothesis can be achieved by generating a plan for the corresponding planning problem. Furthermore, the most plausible hypothesis is a plan with minimum cost. This relationship makes it possible to experiment with a wide variety of planners. However, we first need to address the following questions before experimenting with planners.

Encoding Observations In the modeling section we discussed the form of an observation formula φ we consider for malware detection where all observations are totally ordered. In the corresponding planning problem, we consider these observations as temporally extended goals. However, planners capable of handling temporally extended goals are limited and not advanced enough (Sohrabi, Baier, and McIlraith 2010). Therefore, we need to use a classical planner but first we need to compile observations away. Our approach is similar but uses the simplified version of the encoding proposed in (Haslum and Grastien 2011) because our observations are totally ordered. That is instead of having an "advance" action that ensures the observation order is preserved, each action that emits an observation has an ordering precondition. Hence, only a "pending" observation can be observed, and upon doing so, the next observation becomes pending. This ensures that the generated plans meet the observations in exactly the given order.

Assigning Costs We encode the plausibility notion as actions costs. In particular, we assign a high cost to the *discard* action in order to encourage explaining more observations. In addition, we assign a higher cost to all instances of the actions that represent malicious behaviors than those that represent non-malicious behaviors (this is relative, that is if we assign a cost of 20 to a malicious action instance, then we assign cost less than 20 to all other non-malicious action instances). Also actions that belong to the same lifecycle state can have different costs associated with them. This is because not all actions are equally likely and can have the same weight in determining the root cause. However, we could have additional knowledge that indicates a more plausible root cause. For example, if we have two observations, a download from a blacklisted domain, and download an executable file, we could indicate that an infection is more likely if downloading from a blacklisted domain. This can be achieved by assigning a higher cost to the action that represents infection by downloading an executable file than the action that represents downloading from blacklisted domain.

Using Replanning to Generate Plausible Hypotheses Given a description of a planning problem, we need to generate multiple high-quality (or, equivalently, the low-cost) plans, each corresponding to a distinct plausible hypothesis.

0\. Find plan P for the original problem.
1\. Add each action a of P as a separate action set $S=\{a\}$ to future exploration list L.
2\. For each set of actions S in L,
3\.   For each action $a \in S$,
4\.   Add negated predicate associated with a to the goal.
5\.   Generate a plan P for the new problem where the goal disallows all actions in S.
6\.   For each action $a \in P$,
7\.   Add the set $S \cup \{a\}$ to L'
8\. Move all action sets from L' to L.
9\. Repeat from step 2.

Figure 2: Replanning algorithm

It should be noted that there is recent effort in generating diverse plans (e.g., (Srivastava et al. 2007)) but rather than diverse plans, we want to find a set of high-quality plans. However, we are not aware of any existing planner capable of generating a set of near-optimal plans. To overcome this, we have developed a replanning-like approach that allows us to use any planner that can generate a single optimal or near-optimal plan quickly. In our experiments, we exploit LAMA (Richter and Westphal 2010), the first place winner of the sequential satisficing track in the International Planning Competition (IPC) 2008 and 2011 for this purpose.

We have extended the planning domain by associating a new unique predicate with each action, and including every parameter of the action in the predicate. By adding this predicate to the effect of the action, and its negation, with specific parameter values, to the goal, we can prevent a specific instance of an action from appearing in the plan.

To drive the replanning process, we have implemented a wrapper around the planner. The algorithm for the wrapper is outlined in Figure 2. The wrapper first generates an optimal or near-optimal plan using an unmodified problem. It then modifies the problem to exclude the specific action instances of the first plan one by one, and generates a near-optimal plan for each modification. The wrapper then recursively applies this procedure to each plan from the new set, this time excluding both the action from the new plan, and the action that was excluded from the first plan when generating the new plan. The process continues until a preset time limit is reached. Separately, a time limit can be specified for each planner invocation (each iteration of the planner), to ensure a minimum of modified goals explored. In general the number of excluded actions is equal to the depth of the search tree the algorithm is traversing in a breadth-first manner. Our replanning algorithm can eventually find every valid plan if used with a complete planner and given enough time.

In our implementation, we only use a subset of actions sufficient for generating different plans. We also sort action sets in step 2, for example to ensure the instances of the *discard* action are removed before other actions. Finally, we save all plans, if multiple plans are generated by the planner during one round. However, we only use one best generated plan to generate new action sets.

This process generates multiple distinct plans, and therefore hypotheses. After sorting them by cost, a subset can be presented to administrators or automated systems as possible hypotheses for future investigation.

Experimental Evaluation

When the presence of malware is detected inside critical infrastructure, the network administrators have little time to investigate the threat and mitigate it. Thus, malware must be discovered as quickly as possible. However, accuracy is no less important. Critical infrastructure disruptions resulting from overreaction to suspicious-looking observations can be just as undesirable as malware infections themselves.

The experiments we describe in this section help evaluate the response time and the accuracy of our approach.

The lifecycle models need to include descriptions of both non-malicious and malicious behaviors, and may need to be modified regularly to match changes in network configuration and knowledge about malware threats. To study the performance of our approach on a wide variety of different lifecycles, we have generated a large set of lifecycles randomly. In generating the lifecycles, we have ensured that a directed path exists between each pair of states, 60% of the states are bad, 40% are good, and both unique and ambiguous observations can be associated with states. In addition, we have also evaluated performance by using a hand-crafted description of the lifecycle shown in Figure 1.

Planning Domain Model The planning problems are described in Planning Domain Definition Language (PDDL) (McDermott 1998). One fixed PDDL domain including a total of 6 actions was used in all experiments. Actions *explain-observation* and *discard-observation* are used to advance to the next observation in the sequence, and actions *state-change* and *allow-unobserved* change the state of the lifecycle. Two additional actions, *enter-state-good* and *enter-state-bad*, are used to associate different costs for good and bad explanations. In our implementation, the good states have lower cost than the bad states: we assume the observed behavior is not malicious until it can only be explained as malicious, and we compute the plausibility of hypotheses accordingly. The state transitions of malware lifecycle and the observations are encoded in the problem description. This encoding allowed us to automatically generate multiple problem sets that include different number of observations as well as different types of malware lifecycle.

Performance Measurements To evaluate performance, we introduce the notion of *ground truth*. In all experiments, the problem instances are generated by constructing a ground truth trace by traversing the lifecycle graph in a random walk. With probability 0.5 the ground truth trace contained only good states. For each state, a noisy or missing observation was generated with probability 0.025, and ambiguous observations were selected with probability 0.25.

Given these observations, each of the generated plans represents a hypothesis about malicious or benign behavior in the network. We then measure performance by comparing the generated hypotheses with the ground truth, and consider a problem *solved* for our purposes if the ground truth appears among the generated hypotheses.

For each size of the problem, we have generated 10 problem instances, and the measurements we present are averages. The measurements were done on a dual-core 3 GHz Intel Xeon processor and 8 GB memory, running 64-bit RedHat Linux. Each single LAMA invocation was allowed to run up to 20 seconds, except for the first invocation, and all plans produced in that time were saved. Replanning iterations were repeated until the 300 seconds time limit was reached. We did not set a time limit for the first invocation (i.e., the first invocation can take up to 300 seconds). This is because we wanted LAMA to find the optimal or near-optimal plan (by exhausting the search space) in the first invocation before starting to replan. In the harder problems, if the first invocation did not finish, no replanning was done.

As expected, our approach results in many replanning rounds that together help produce many distinct plans. This can be seen in Table 1, showing the average total number of replanning rounds in the *Total* column, the average number of unfinished rounds that were terminated due to per-round time limit in the *Unfinished* column, and the average number of distinct plans at the end of iterations in the *Plans* column. Note, we only count distinct plans, independent subtrees in the iterative replanning process may produce duplicates. Also in the smaller size problems, more replanting rounds is done and hence more distinct plans are generated which increases the chance of finding the ground truth.

In both Table 1 and Table 2, the rows correspond to the number of generated observations and the columns are organized in four groups for different lifecycle types. The hand-crafted lifecycle contained 18 lifecycle states and was not changed between the experiments. The generated lifecycles consisted of 10, 50 and 100 states and were re-generated for each experiment, together with the observations.

Table 2 summarizes the quality of the plans generated in these experiments. The *% Solved* column shows the percentage of problems where the ground truth was among the generated plans. The *Time* column shows the average time it took from the beginning of iterations (some finished and some unfinished rounds) to find the ground truth solution for the solved problems. The dash entries indicate that the ground truth was not found within the time limit.

The results show that planning can be used successfully to generate hypotheses for malware detection, even in the presence of unreliable observations, especially for smaller sets of observations or relatively small lifecycles. However, in some of the larger instances LAMA could not find any plans. The correct hypothesis was generated in most experiments with up to 10 observations. The results for the hand-crafted lifecycle also suggest that the problems arising in practice may be easier than randomly generated ones which had more state transitions and higher branching factor.

	Hand-crafted	10 states	50 states	100 states
Solved in 1st round	56	18	8	3
Solved in rounds 2-50	4	12	5	3
Not solved	20	50	67	74

To assess the impact of iterative replanning on solution accuracy, in the same experiments, we have collected the information about the number of iterations required to find the ground truth solution. The above table summarizes the results. The ground truth is found quickly in several cases by LAMA in the first iteration. In some experiments additional iterations helped find the ground truth. But note that there

	Hand-crafted			10 states			50 states			100 states		
Observations	Plans	Total	Unfinished	Plans	Total	Unfinished	Plans	Total	Unfinished	Plans	Total	Unfinished
5	55	261	0	75	340	0	50	130	0	40	49	0
10	80	176	0	128	248	0	82	78	0	32	20	1
20	117	111	0	156	171	0	52	34	0	4	14	13
40	78	58	0	105	120	0	18	13	11	4	2	2
60	42	36	0	81	81	0	5	10	9	3	1	1
80	30	21	4	49	38	0	4	2	2	2	1	1
100	25	16	8	36	28	3	3	1	1	0	1	1
120	20	14	12	30	28	5	2	1	1	0	1	1

Table 1: The average number of LAMA replanning rounds (total and unfinished) and the number of distinct plans generated.

	Hand-crafted		10 states		50 states		100 states	
Observations	% Solved	Time	% Solved	Time	% Solved	Time	% Solved	Time
5	100%	2.49	70%	0.98	80%	5.61	30%	14.21
10	100%	2.83	90%	2.04	50%	25.09	30%	52.63
20	90%	12.31	70%	24.46	-	-	-	-
40	70%	3.92	40%	81.11	-	-	-	-
60	60%	6.19	-	-	-	-	-	-
80	50%	8.19	-	-	-	-	-	-
100	60%	11.73	10%	10.87	-	-	-	-
120	70%	20.35	20%	15.66	-	-	-	-

Table 2: The percentage of problems where the ground truth was generated, and the average time spent for LAMA.

are still many cases where the ground truth solution was not found within the time limit.

Overall, iterative replanning proposed in this paper helped increase the number of solved problems, and generate valuable hypotheses, which would otherwise have been missed. However, when the size of the plan increases, our approach becomes less effective too, requiring exponentially greater number of iterations to reach the same depth in the breadth-first search. Further, a real-world malware attack may try to deliberately hide ground truth among plausible hypotheses. Hence, we believe that to fully address this problem, new planning algorithms must be able to find multiple near-optimal plans efficiently.

Summary and Discussion

Several researchers have looked into addressing the cyber-security problem by means of planning (Boddy et al. 2005; Lucngeli, Sarraute, and Richarte 2010; Roberts et al. 2011), our approach and focus however is different. In particular, rather than finding "attack plans" for penetration testing purposes we are focused on generating plausible hypotheses that can explain the given set of observations which are the result of performing analysis of network traffic. The generated hypotheses can then be used by an automated system or the network administrator to do further investigations.

There are several approaches in the diagnosis literature related to ours in which use of planners as well as SAT solvers are explored (e.g., (Grastien et al. 2007; Sohrabi, Baier, and McIlraith 2010)). In particular, the work on applying planning for the intelligent alarm processing application is most relevant (Bauer et al. 2011; Haslum and Grastien 2011). Similarly, they also considered the case where they can encounter unexplainable observations but did not give any formal description of what these unexplainable observations represent and how the planning framework can model them. Furthermore, it is not clear how their *non-exhaustive* approach can find the ground truth or "what really happened" by simply finding a single plan. On the other hand, the *exhaustive* approach to the diagnosis problem where all diagnoses that are consistent with the observations are found is also relevant (e.g., (Grastien, Haslum, and Thiebaux 2011)). However, observations were assumed to be reliable, while our focus was to address both unreliable observations and finding multiple high-quality plans in a single framework. It may be possible to extend the exhaustive approach using our techniques to address unreliable observations.

In this paper we addressed the hypothesis exploration problem for malware detection using planning. To that end, we proposed a characterization of the hypothesis generation problem and showed its correspondence to an AI planning problem. Our model incorporates the notion of hypothesis plausibility which we map to plan quality. We also argued that under unreliable observations it is not sufficient to just find the most plausible hypothesis for effective malware detection. To generate high-quality plans (not necessary the top high-quality plans) we proposed a method that enables a planner (in our case the LAMA planner) to run in an iterative mode to further explore the search space in order to find more plans. Our results show that running LAMA repeatedly with modified goals can improve the chance of detecting the ground truth trace. However, there are still cases where the ground truth trace cannot be found by the planner.

We believe that LAMA would have had a better chance of detecting the ground truth trace if instead of finding a set of high-quality plans it could have generated the top k plans, where k could be determined based on a particular scenario. In particular, although searching for the optimal plan, or finding a set of sub-optimal plans is interesting and challenging, the malware detection application we looked at in this paper is an example that shows this may not be sufficient. We hope that the results presented in this paper can inspire the planning community to develop algorithms and techniques capable of generating not only a single optimal plan but also the top high-quality plans.

References

Bauer, A.; Botea, A.; Grastien, A.; Haslum, P.; and Rintanen, J. 2011. Alarm processing with model-based diagnosis of discrete event systems. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX)*, 52–59.

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 12–21.

Cassandras, C., and Lafortune, S. 1999. *Introduction to discrete event systems*. Kluwer Academic Publishers.

Cordier, M.-O., and Thiébaux, S. 1994. Event-based diagnosis of evolutive systems. In *Proceedings of the 5th International Workshop on Principles of Diagnosis (DX)*, 64–69.

Emerson, E. A. 1990. Temporal and modal logic. *Handbook of theoretical computer science: formal models and semantics* B:995–1072.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 81–88.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI)*, 305–310.

Grastien, A.; Haslum, P.; and Thiebaux, S. 2011. Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX)*, 60–67.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: Two case studies. In *International Scheduling and Planning Applications woRKshop (SPARK)*, 27–44.

Keyder, E., and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *Journal of Artificial Intelligence Research* 36:547–556.

Lucngeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In *Workshop on Intelligent Security (SecArt 2010)*.

McDermott, D. V. 1998. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

McIlraith, S. 1994. Towards a theory of diagnosis, testing and repair. In *Proceedings of the 5th International Workshop on Principles of Diagnosis (DX)*, 185–192.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1778–1783.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Roberts, M.; Howe, A.; Ray, I.; Urbanska, M.; Byrne, Z. S.; and Weidert, J. M. 2011. Personalized vulnerability analysis through automated planning. In *Working Notes of IJCAI 2011, Workshop Security and Artificial Intelligence (SecArt-11)*.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 26–36.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, 261–267. Accepted as both oral and poster presentation.

Srivastava, B.; Nguyen, T. A.; Gerevini, A.; Kambhampati, S.; Do, M. B.; and Serina, I. 2007. Domain independent approaches for finding diverse plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016–2022.

Thorsley, D.; Yoo, T.-S.; and Garcia, H. E. 2008. Diagnosability of stochastic discrete-event systems under unreliable observations. In *Proceedings of American Control Conference*, 1158– 1165.