Automated Vulnerability Analysis Using AI Planning

Steven Harp, Johnathan Gohde, Thomas Haigh, Mark Boddy
Adventium Labs
111 Third Avenue So.
Minneapolis, MN 55401
mark.boddy@adventiumlabs.org

Abstract

As networked systems become more complex, and they support more critical applications, there is a compelling need to augment the Red Team approach to vulnerability analysis with more formal, automated methods. Artificial Intelligence (AI) Planning, with its well-developed theory and rich set of tools, offers an attractive approach. By adopting this approach we have been able to generate attack graphs for a simple but realistic web-based system in five seconds or less, which is an order of magnitude improvement over previous efforts at automated analysis. In this paper we describe our methods and the results. Since vulnerability analysis is a new application of AI planning, our work has uncovered issues with both modeling techniques and planning tools. We discuss these issues and suggest methods for addressing them.

1 Introduction

This paper describes new automated techniques to help defenders understand the weaknesses of their systems against various kinds of potential attackers. In our Behavioral Adversary Modeling System (BAMS), the security analyst selects the attributes and objectives of a model adversary and then uses Artificial Intelligence (AI) planning algorithms to automatically generate the courses of action (COAs) that such an adversary is likely to choose in subverting the system. The analyst can then use this information to evaluate the susceptibility of his system to attacks by a selected adversary, and to select the most reasonable defensive measures.

Understanding an attacker's strategic goals, available resources, tolerance for risk, and tactics successfully used in the past makes it possible, in theory, to predict the enemy's likely COAs in various situations. The value here is in the predictive behavioral model of the enemy, not in the collection of enemy characteristics and preferences. Most models, such as psychological profiles or analysis of past tactics, are only useful when they can somehow be transformed into realistic, predictive behavioral models. Defenders need models that provide realistic predictions of specific adversary behaviors so they can concentrate on defending against the attacks their adversaries are likely to execute rather than simply organize their defenses to protect against the latest published attacks.

We have implemented this technique in a prototype behavioral adversary modeling system, BAMS. Our work was motivated by a particularly challenging problem: understanding the threat posed by malicious insiders (MIs), adversaries who are legitimate users of the system, with the ability to mount physical, cyber, and social engineering exploits to achieve their goals. The BAMS prototype has been used to anticipate insider attacks on document management systems (DMSs). The prototype has given excellent results on a test system consisting of a web-based DMS. The system can generate a sequence of attacks against the same network, exploiting different vulnerabilities as old ones are denied by changes to the configuration. The plans generated have varied from twenty to fifty atomic steps, and are produced rapidly enough (typically less than a second) to make the interactive use of the tool attractive.

2 Vulnerability Analysis as a Planning Problem

Several recent attempts to construct descriptive adversary models in terms of profiles have been described. For example a descriptive model of malicious insiders developed by Wood appears in an appendix to a Rand report [1]. The elements in Wood's profile include an adversary's motivation, technical skills, system access, knowledge, privilege, and tolerance for risk. He proposes a similar model for cyber-terrorists and identifies some DARPA Red Team exercises where the model could be applied to assist the Red Team in exhibiting

realistic behavior. In essence, this approach maps a previously identified attack and an adversary description to a boolean value that indicates whether or not the adversary could or would mount the attack.

In our view, this is the wrong direction for domains as fluid as these. Descriptive adversary models do not meet the needs of system and network security analysts. They need models that provide insight into the sort of attacks that they can expect to see from the classes of adversaries most likely to attack their systems. It makes more sense to map from a description of adversary characteristics to the set of attacks that the adversary is capable of or is likely to try. Rather than hypothesize or collect attacks to test against an adversary model, we use the model to derive possible attacks. This helps to isolate the process from the blinders of defender biases.

Given this sort of a Behavioral Adversary Model, analysts can make intelligent trades among different security architectures and counter-measures. They can also use these models to assess the risk associated with trusting the system to protect mission critical data or services. Red Teams can use behavioral models to develop more realistic attack scenarios than the ones they use today, ideally capturing the preferences and tendencies of particular classes of adversaries in question, rather than the skill set or preferences of the red team. These models must address the full range of physical and social engineering exploits as well as the cyber exploits that an adversary might employ.

One way to build behavioral adversary models is to use an adversary description as an input and use an automated process to generate the COAs that the adversary is likely to follow in attacking the system being studied. This requires an efficient, highly scalable method of generation. It also requires tools for quickly reconfiguring a problem so that the analyst can explore "what-if" scenarios to determine the impact of different counter-measures, or even the impact of an hypothesized zero-day attack. We have applied planning techniques from AI to achieve this.

2.1 AI Planning

Figure 1 illustrates the framework for AI planning. The planner takes as input an initial state, a set of operators defining possible actions, and a goal state. As output, it produces a legal plan (or plans) as a sequence of operations, or indicates that no plan leading from the initial state to the goal is possible. The actions encoded by operators move the world from one state to another.

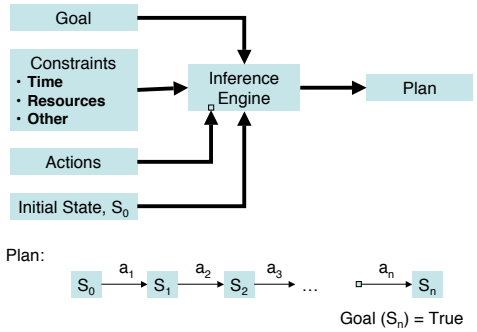

Figure 1: The AI planning process.

Additionaly, logical or metric constraints such as monetary cost or cumulative detection risk, can be used to limit the space of legal plans or rank one plan as preferrable to another.

Our approach creates a rich adversary model and uses it as input to an automated process for generating attacker COAs. This approach requires a translation from the set of adversary characteristics described by Wood into sets of facts and exploits in the formal model.

One concern evident from the beginning has been choosing the best level of abstraction at which to model the target system. If too abstract, the resulting COAs tend to be uninteresting, e.g. two-step plans like: "the attacker gains root privileges on the server and then reads the secret document." At too fine a level, (e.g. network packets or individual system calls) the plans are full of uninteresting details. We have aimed for an intermediate level of abstraction that yields plans that vary from a dozen to a few dozen steps.

2.2 Modeling the Environment

The current BAMS prototype uses over 25 different classes of objects, including the most basic elements of the computational environment: hosts, processes, programs, files, user identifiers (uids), group identifiers (gids), email messages, cryptographic keys, network switches, firewalls, and keystroke logging devices. The model also includes physical types, such as people, rooms, doors, locks, and keys. There are also types to describe information concepts such as secrets, and task instructions that may be sent from one person to another.

A variety of facts about the objects can be listed to define the state of the world that is being reasoned about. These facts are represented with relations (formally "predicates"), and are used to represent the initial state of the problem, the goal state, and the inter-

mediate states as well. Some of these facts represent conditions that may change as a result of the execution of the plan. For example, an MI may move from his office to some other room. Other facts are static, e.g. the program `ms_outlook` will continue to have the capability of reading email, and the skill levels of the attacker may not change during an attack. Our current prototype has 124 predicates, and a typical problem will use these to express 200-300 facts.

In BAMS, an adversary's objectives are the goals which are presented to the planner. The goals can consist of propositions which need to be true as well as constraints for metric information. For instance, a goal for our canonical MI "bob" may be to minimize the detection risk of the plan while gaining access to a secret document, such as:

```
(:goal (knows bob secret_info))
(:metric minimize (detection_risk))
```

The goal could also have a hard constraint such as keeping the detection risk below a certain level as is the case here:

```
(:goal (and (knows bob secret_info)
            (<= (detection_risk) 5)))
```

The means with which the adversary is able to achieve the goals are the operators which are encoded into the domain. These operators represent the primitive actions which can be taken, such as entering a username or selecting a recipient for an email. Our current prototype uses 56 operators. Each of these operators is composed of a set of preconditions, an action description, and a set of effects. An example of an operator describing selection of an email recipient is provided in Figure 2.

Operators usually are defined with one or more typed parameters, i.e. formal arguments. These variables are prefixed with a question mark in the PDDL syntax. For an operator to be applied, the variables must be instantiated with real objects of the correct types in the domain, and the facts stated in the preconditions must be present in the plan at the time the operator is to be used. After the operator is applied, the facts stated in the effect clause are asserted to be true. The expressions may include logical connectors (and, or, not, implies) and quantifiers (exists, for-every). The process of instantiating and checking is handled automatically by the planning algorithm.

We have added two enhancements to isolate users of BAMS from the need to directly represent facts and operators in PDDL. The first is a set of M4 macros that encapsulate some of the language's syntactic conventions and augment them, e.g. with explanatory text (used in presenting discovered plans). Macros also support the notion of "modal" sequences of actions, which efficiently capture certain common state-machine-like situations such as the assembling and sending of an email message, or the launching of a new operating system process. An example of a macro, `defmodal`, is given in Figure 2. The final problem and operator definitions presented to the planner are assembled dynamically by M4 and a make script from component modules. The modularity allows a separate file to contain all the predicates and classes pertaining to a single domain, e.g. email.

```
defmodal(senderset, senderset,
 ADD_RECIP
  :parameters (?sender - c_human
               ?email - c_email
               ?ruid - c_uid
               ?recip - c_human)
  :precondition
  (and (writing_email ?sender ?email)
    (or (trusts_recipient ?sender ?recip)
        (insider ?sender))
        (has_uid ?recip ?ruid))
  :effect
  (and (recipient ?email ?ruid)),
  "?sender adds recipient
   ?recip (?ruid) to ?email")
```

Figure 2: Definition for operator ADD_RECIP, which adds a recipient to an email.

The second enhancement is provided by a BAMS User Interface (BUI). With it, the particulars of the problem are presented in a set of tabbed dialogs. In Figure 3, a simulated office layout, with rooms, doors and locks can be graphically manipulated by dragging and clicking, to situate the participants in more or less dangerous situations. A snapshot from a guided dialog sequence (wizard) to create a profile for a new malicious insider is shown in Figure 4.

The user selects the elements and options to present to the planner, and then issues a "RUN" command from the menu. The correct PDDL is generated, the planner run, and the output translated and displayed in the interface.

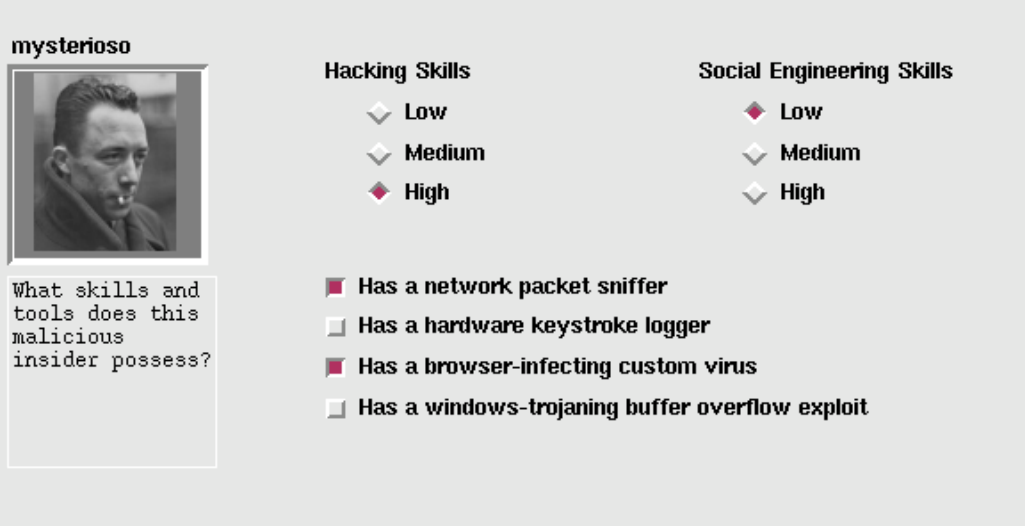

Figure 4: Part of a dialog sequence for specifying a new malicious insider.

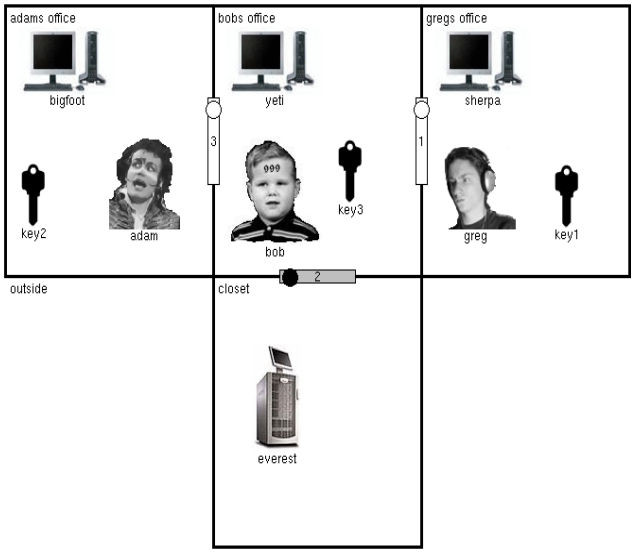

Figure 3: Top level screen of BAMS console.

2.3 Planning Algorithms

An advantage of using AI planners is that they produce a course of action in a fully automated way. The planner synthesizes a plan by stringing together a series of operators in an order which provably achieves the goals. These plans are produced at a level of detail that is found in the model. The BAMS model has actions represented down to the level of entering information for logging into a system or selecting a recipient for an email. This is a finer level of detail than has been traditionally used for generating attack trees. It is possible to add further detail by modifying the model alone. The planning tool would not require any changes in order to produce courses of action from the new model. This makes the BAMS approach easily extensible when new exploits are discovered. Abstract actions may be introduced into the model to allow the effects of potential future exploits to be explored. For example, an analyst may introduce an action that subverts a certain suid daemon without any particular vulnerability in that daemon yet known. This operation can then be used in producing plans allowing the analyst to see what an attacker could make of such a vulnerability.

Since the synthesis of plans proceeds the unordered set of possible actions, there is no preformed notion of how a sequence of actions must be performed. For instance, the BAMS model represents that a user cannot send an email without being logged into a computer, but places no restrictions on when an email might be sent if the conditions for sending email are met. This enables the system to possibly discover novel COAs, i.e. ones that do not follow a preformed notion of an attack. If the planner produces a plan, that plan is provably admissible–it is one method of reaching the goals stated in the problem. If no plan is created, then there is provably no method in the model with which the problem can be solved. In some instances, but not all, it would require an exhaustive search to determine there are no plans.

The planner that was used in this system is Jörg Hoffmann's Metric-FF [4] planner (hereafter FF). FF (Fast Forward) is a forward searching planner which uses a

relaxation of the planning network pioneered in Graphplan [2] to provide heuristic estimates of the distance of a state to the goal. These heuristic estimates are then used to guide search. With FF there are two different search modes, enforced hill climbing (EHC) and best first search (BFS). EHC is the default initial search mode, and can be very fast, but does not always find a solution when one exists. If EHC fails to find a solution then the slower but complete BFS mode is used. If BFS does not return a solution then it is provably true that no solution exists.

EHC, like other hill-climbers, gains speed by focusing on one path rather than attempting to find which action upon many possible paths would lead to the goal. Along the single path, FF picks the most promising looking action and chooses that. In most cases this method is sufficient to find a goal state. BFS is more similar to an A* search, expanding along many different potential courses of action. This differentiation in action selection is responsible for improvements in speed as well as completeness.

We have experimented with a wide variety of planners of this class, and FF has been the best planner for the needs of this system so far for two reasons. First, it is fast enough to handle problems of an interesting size and complexity. (FF has won several awards in the International Planning Competition [7]). Second, FF is thorough in its handling of the PDDL modeling language–all of the features we need are supported, including metrics, and the use of complex expressions in the preconditions and effects of actions. Memory usage of FF on our problems is also acceptable, although we have created a modified version that is more parsimonious than the original source. Because this sort of planner creates extensive data structures prior to any search, it is important to keep these in physical RAM if at all possible for the sake of speed.

2.4 Results

As one test of the system, we created a problem environment which had four hosts, a firewall separating two network segments, a malicious insider, a normal user, and an administrator. Several vulnerabilities were present in this configuration, including hosts with no virus scanning, a non-switched network, and trusting coworkers. The system was run on this problem with a goal for the MI to extract secret information that was stored in a document in the DMS. A typical plan is shown in Figure 5.

Illustrating one use of this system, the initial environment was incrementally repaired. After a plan was found, a minimal number of changes were made to the problem to remove a particular vulnerability that was exploited, and the planner was given another opportunity to plan with the modified environment. This process was repeated until no more plans were found. Table 1 shows the plan lengths (in steps) and time to generate each for this sequence. The titles are interpretations that we provided to the synthesized plans. Of interest is that the length of the plans vary but the time taken to generate the plans remains low. The timing for most of these plans is dominated by factors other than pure search. The plan which took the longest (sniffing a password from an email) fell back to the use of the BFS algorithm, and search time was a significant contributor to overall time.

3 Related Work

Our approach contrasts with hierarchical styles of planning, in which high level goals are recursively decomposed into subgoals, eventually terminating with primitive actions. This is conceptually similar to methods widely used for attack tree generation, e.g. an initial abstract plan might be: “first, scan the hosts; next, get root; then, install a backdoor”. This would then be expanded, e.g. by exploring different methods for getting root. This technique can be efficient, but also tend to introduce biases and blind spots, since they closely reflect the way a particular author has conceived of attacks. Ideally, the automated system could “think outside the box”. For this reason, we believe that constructive planners, like we have used, have greater long term potential to produce novel and unanticipated plans.

Other researchers have used automated methods to perform various sorts of vulnerability analysis. In [12] Wool describes a firewall simulator that takes as input a firewall’s configuration and routing tables and returns a description of the firewall’s response to a wide variety of network packets. Zerkle and Levitt [13] developed the NetKuang system to check for misconfiguration on Unix networks. They do this by examining the configuration files on all the hosts and employing a brute force backward search from the goal to generate an attack graph if it exists. At NSPW in 1998, Phillips and Swiler first proposed using automatically generated attack graphs to perform a general vulnerability analysis for a computer network [9]. They illustrate their approach with a simple network and a small set of exploits, but they provide little insight into how to actually generate the graphs.

```
0 : ADAM sits down at BIGFOOT
1 : ADAM enters ADAM_UID as user name for login on host BIGFOOT
2 : ADAM enters password ADAM_PWD for login at host BIGFOOT
3 : Shell B_WEXPLORE is launched on host BIGFOOT for user ADAM_UID
4 : Program WEXPLORER on host BIGFOOT forks a child process
5 : Contents of file B_IEXPLORE begin executing as uid ADAM_UID on host BIGFOOT
6 : BOB sits down at YETI
7 : BOB enters BOB_UID as user name for login on host YETI
8 : BOB enters password BOB_PWD for login at host YETI
9 : Shell Y_WEXPLORE is launched on host YETI for user BOB_UID
10 : Program WEXPLORER on host YETI forks a child process
11 : Contents of file Y_ETHEREAL begin executing as uid BOB_UID on host YETI
12 : ETHEREAL starts sniffing the networks on YETI
13 : ADAM logs onto dms admin server EVEREST from BIGFOOT
14 : BOB reads the sniffer thus learning NES_ADMIN_PASS
15 : Program WEXPLORER on host YETI forks a child process
16 : Contents of file Y_IEXPLORE begin executing as uid BOB_UID on host YETI
17 : BOB logs onto dms admin server EVEREST from YETI
18 : DMS session DMSS1 has begun
19 : BOB begins a DMS session on YETI
20 : Connect DMS session DMSS1 to server NES on EVEREST
21 : A route from YETI to DMS server EVEREST exists
22 : BOB enters password BOB_DMS_PWD for the DMS session.
23 : Authenticate BOB_UID in dms session DMSS1 with EVEREST using BOB_DMS_PWD
24 : BOB adds an acl to allow read access of E_SECRET_DOC to the EAST_GID group
25 : BOB begins a DMS request at YETI in session DMSS1
26 : Document E_SECRET_DOC is requested in session DMSS1
27 : Document E_SECRET_DOC is sent and displayed on YETI in session DMSS1
28 : BOB reads E_SECRET_DOC and learns SECRET_INFO
```

Figure 5: An automatically generated attack plan.

Ritchey and Ammann propose a model checking approach [10]. They model the connectivity of the host systems and their vulnerabilities as well as the exploits available to the attacker. Then they put the model checker to work trying to generate a counter-example to a selected security property. The counter-example takes the form of an attack graph. Sheyner et al. have extended the model-checking approach [11]. They have developed techniques for identifying a minimal set of security counter-measures to guarantee a safety property, and to compute the probability of attacker success associated with each attack graph.

None of these approaches have claimed particularly good performance. For example, Sheyner et al. illustrate their approach with a simple network model consisting of a generic outside intruder separated from a two server network by a simple firewall. They report that in a case where the attacker had eight exploits at his disposal, it took two hours to generate the attack graph. Also, since the focus of these approaches has been vulnerability analysis, they do not put much emphasis on attacker modeling.

The work on "Network Hardening" by Noel et al. [8] is very close to ours in approach. Their "exploit graph" is quite similar to, though apparently developed independently from, work in AI on Plan Graphs, which form the basis for the planning system employed here [3].[1] Their assumption of nonmonotonicity in an attacker's access to the system corresponds to one of the simpler of a large family of "tractable subclasses" of AI planning (see, for example, [5]).

Noel et al.'s results regarding the computation of minimal network hardening options is a goal of our work as well. While they have gone farther in the computation of these minimal hardening sets, the assumptions they make to achieve efficient computation of those sets are quite restrictive, ruling out the representation of several classes of model features. For example, among

[1] In particular, both approaches focus on pre- and post-conditions, and gain exponential savings over the unfactored state-space approaches.

Description	Time to Generate	Steps in Plan
Direct Client Hack	0.67 seconds	25 Steps
Misdirected Email	0.67 seconds	32 Steps
Shoulder Surfing	0.69 seconds	18 Steps
Email Trojan	0.71 seconds	37 Steps
Spoofed Email Trojan	0.73 seconds	37 Steps
Spoofed Instructions	0.79 seconds	36 Steps
Administrator ACL Change	1.20 seconds	23 Steps
Sniff Administrator Password	1.62 seconds	28 Steps
Sniff Password from Email	4.77 seconds	44 Steps

Table 1: Time to Generate and Length of Each Plan of Attack

the implications of a monotonic model is not only that access once gained is never lost, but also that there are no mutually-exclusive ways to gain a certain access. In addition, there are numerous expressive extensions in current planning systems which support complex models in a computationally-efficient way, including quantified preconditions and effects, context-dependent effects (a feature which we have used extensively), and parameterized operators, permitting much more efficient representation and reasoning over a large universe of objects (hosts, programs, users, *etc.*).

4 Further Research

Our research shows that AI planning techniques can be adapted to generate adversary COAs. We have developed a modular approach to modeling and a user interface that a non-specialist can use to rapidly compose pieces of the model to create new adversary profiles and reconfigure the problem's physical and cyber space. To validate the approach, we have created a Proof of Principle demonstration involving insider attacks on a simple, yet realistic web-based document management system. Our tools can generate reasonably complex attacks of twenty to fifty steps in a few seconds.

While a major step forward in automated vulnerability analysis, the BAMS prototype is some distance from being a general purpose off-the-shelf solution. In the course of our research we have identified further requirements for future planners and modeling approaches in this domain.

Efficient Multiple Plan Generation It would be useful to generate in batch mode, all of the plans to achieve a given attacker goal, thus enabling comparative analysis. Modern AI planners build some of the data structures required to generate multiple plans, but few of them exploit this to actually generate them. One of the unaddressed difficulties is generating multiple plans that are different in *interesting* ways. For example, two plans that differ in the particular process identifier used to launch a program are fundamentally the same. Finding equivalence classes of plans that differ in their functional mechanisms is an open research topic.

Plan Bottleneck Analysis Currently, BAMS works interactively with a security analyst who interprets the plans. A further level of automation would be a sort of "bottleneck analysis" to automatically identify the critical steps that enable a set of COAs. This would point to a particular place to concentrate defenses. The capability may also facilitate the efficient generation of multiple plans.

Performance and Scaling The predicates used by the model, the set of objects in the problem, the complexity of the operators, the form of the goal and constraints, and the algorithms used by the planner interact to determine the time and space required to find plans. Performance deteriorates when large numbers of objects are added or when operators with many parameters are inserted. While we know how to ameliorate these effects in specific cases, techniques to handle these issues in a more general way must be added to address larger problems.

Probabilistic Planning There is no notion of uncertainty or likelihood in the current plans generated by BAMS: a plan is possible for the attacker or not. Yet, for a given attacker, some plans will definitely be more likely than others. Augmenting descriptions of plans with something like the joint probability of the actions is a natural extension we have begun to explore.

Model Generation Tools Currently, extending the model to include new operations and predicates can require planning expertise. It has not yet been reduced to point and click operations in the user interface. To this end, methods to axiomatize domains, such as action languages [6] have been explored in a preliminary way.

Implementing this without losing performance may entail solving some subtle problems in logic in an offline compilation step. Constructing models of existing systems by importing network maps or scans is also a possibility.

Our ongoing work will address these open research challenges.

Acknowledgments

This work was supported by ARDA under contract NBCHC030080.

References

[1] R. H. Anderson, R. Brackney, and T. Bozek, "Advanced Network Defense Research," Technical report, National Defense Research Institute, RAND, 2000.

[2] A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pp. 1636–1642, 1995.

[3] J. Hoffmann, "The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables," *Journal of Artificial Intelligence Research*, 2003. accepted for the special issue on the 3rd International Planning Competition.

[4] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[5] P. Jonsson and C. Backstrom, "State-Variable Planning Under Structural Restrictions: Algorithms and Complexity," *Artificial Intelligence*, vol. 100, no. 1-2, pp. 125–176, 1998.

[6] F. Lin, "Compiling Causal Theories to Successor State Axioms and STRIPS-Like Systems," *Journal of Artificial Intelligence Research*, vol. 19, pp. 279–314, 2003.

[7] D. Long and M. Fox, "The 3rd International Planning Competition: Results and Analysis," *Journal of AI Research*, vol. 20, pp. 1–59, 2003.

[8] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs," in *Proceedings of 19th Annual Computer Security Applications Conference*, pp. 86–95. IEEE Computer Society, 2003.

[9] C. Phillips and L. P. Swiler, "A Graph-Based System for Network-Vulnerability Analysis," in *Proceedings of the New Security Paradigms Workshop*, pp. 71–79, 1998.

[10] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Proceedings 2000 IEEE Computer Society Symposium on Security and Privacy*, pp. 156–165, May 2000.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *2002 IEEE Symposium on Security and Privacy (SSP '02)*, pp. 273–284, Washington - Brussels - Tokyo, May 2002, IEEE.

[12] A. Wool, "Architecting the Lumeta firewall analyzer," in *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001, USENIX.

[13] D. Zerkle and K. Levitt, "NetKuang–A Multi-Host Configuration Vulnerability Checker," in *Proc. of the 6th USENIX Security Symposium*, pp. 195–201, July 1996.