Course of Action Generation for Cyber Security Using Classical Planning

Mark Boddy and **Johnathan Gohde** and **Tom Haigh** and **Steven Harp**[*]
Adventium Labs
Minneapolis, MN
{*firstname.lastname*}@adventiumlabs.org

Abstract

We report on the results of applying classical planning techniques to the problem of analyzing computer network vulnerabilities. Specifically, we are concerned with the generation of *Adversary Courses of Action*, which are extended sequences of exploits leading from some initial state to an attacker's goal. In this application, we have demonstrated the generation of attack plans for a simple but realistic web-based document control system, with excellent performance compared to the prevailing state of the art in this area.

In addition to the new capabilities gained in the area of vulnerability analysis, this implementation provided some insights into performance and modeling issues for classical planning systems, both specifically with regard to METRIC-FF and other forward heuristic planners, and more generally for classical planning. To facilitate additional work in this area, the domain model on which this work was done will be made freely available. See the paper's Conclusion for details.

Introduction

This paper describes our experiences in implementing an automated system intended to aid computer network administrators in analyzing the vulnerabilities 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 potential adversary and then invokes a *Course Of Action* (COA) generator to hypothesize attacks that such an adversary is likely to choose in attempting to subvert the system. The analyst can then use this information to evaluate the susceptibility of his system to attacks by a particular type of adversary, and to select the most reasonable defensive measures.

Our work was motivated by a particularly challenging problem: analyzing the threat posed by malicious insiders, adversaries who are legitimate users of the system, with the ability to mount physical, cyber, and social engineering exploits to achieve their goals. Our BAMS prototype generates hypothetical insider attacks against a simple but realistic model of a web-based Document Management System. BAMS can be used to generate a sequence of attacks against

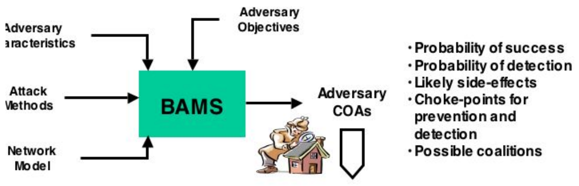


Figure 1: Course of Action Generation from Adversary Models

a given network, exploiting different vulnerabilities as old ones are blocked through configuration changes. The plans generated vary from twenty to fifty atomic steps, and are produced rapidly enough (typically less than a second) to make the interactive use of the tool feasible.

The Course of Action generator is at its heart a classical planner. Most of our work was with Hoffmann's METRIC-FF planner (Hoffmann 2003), for reasons and with results discussed in the body of the paper. In addition to the new capabilities gained in the area of vulnerability analysis, this implementation has provided some insights into performance and modeling issues for classical planning systems, both specifically with regard to METRIC-FF and other forward heuristic planners, and more generally for classical planning.

In the rest of this paper, we present the problem and our implemented solution, including a discussion of modeling issues raised and to some extent resolved, as well as some performance issues encountered in this domain. We then present a brief summary of related work, and a somewhat longer discussion of open questions and future work in this area. To facilitate additional work in this area, the domain model on which this work was done will be made freely available. See the paper's Conclusion for details.

Network Vulnerability Analysis as a Planning Problem

The process we support is the construction and manipulation of *adversary models*, specifically in reasoning from an adversary's goals, capabilities, and knowledge to what they

[*]The work reported in this paper was carried out at Adventium Labs and was supported by ARDA, under contract NBCHC030080. J. Gohde's current affiliation and email: General Dynamics, gohde@cs.umn.edu

may attempt to do in order to achieve those goals. A simple schematic model of this process is presented in Figure 1. The value here is in the predictive behavioral model of the enemy, not in the collection of enemy characteristics and preferences. Unlike most previous work in the area, our system makes predictions of specific adversary behaviors, so that network defenders can concentrate on defending against specific attacks identified as especially likely or especially costly, rather than simply configuring their defenses to protect against the latest published attacks.

Given this *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. To be of any real use, these models must address the full range of physical and social engineering exploits as well as the cyber exploits that an adversary might employ.

Representation

The planning domain and the planning problem are both represented in PDDL (Fox & Long 2002). The current model includes more than 25 different object types, 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 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 might be sent from one person to another.

A variety of facts about these objects can be represented as both fluents (e.g., the current room location of an individual) and static relations (e.g., the physical layout of a building). Our current implementation has 124 predicates, and a typical problem will use these to express 200–300 facts. The predicates denote propositions representing the configuration of hosts, the status of files, the capabilities and vulnerabilities of particular programs, the knowledge possessed by individuals, the dimensions of trust between persons, and the physical layout. Some examples of facts in this domain include:

```
(insider bob)
(in_room bob bobs_office)
(can_unlock key_1 lock_1)
(has_uid bob bob_uid)
(knows bob bob_pwd)
(file_contents_program s_iexplore iexplorer)
(accessible s_iexplore sherpa)
(user_read  s_iexplore greg_uid)
(can_read_email ms_outlook)
(trusts_instructions_by greg adam)
```

Actions are the most complex element of our model. A simple example is the action DMS_ADD_GROUP_ALLOW which modifies the access control list (ACL) for a document by adding group read access for a given group:

```
(:action DMS_ADD_GROUP_ALLOW
   :parameters ( ?admin - c_human
                 ?chost - c_host
                 ?shost - c_host
                 ?doc - c_file
                 ?gid - c_gid  )
   :precondition
   (and (pmode free)
     (nes_admin_connected ?chost ?shost)
     (at_host ?admin ?chost)
     (insider ?admin))
   :effect
   (and
     (dmsacl_read ?doc ?gid)))
```

The precondition (pmode free) is a special construct, discussed in the next section. There are 56 actions in the current domain model.

In BAMS, an adversary's objectives are expressed as goals, which may include both propositional and metric information. For instance, a goal for our canonical bad guy "Bob" may be to minimize the detection risk of the plan while gaining access to a secret document:

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

The goal might also include a hard constraint such as keeping the detection risk below a certain level:

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

The Implemented Model

The domain we model is a simplified Document Management System (DMS), with physical, network, and social environments represented. Figure 2 shows the simulated office layout. The initial contents and state of rooms, doors and locks can be graphically manipulated by dragging and clicking. A snapshot from a guided dialog sequence (wizard) to create a profile for a new malicious insider is shown in Figure 3.

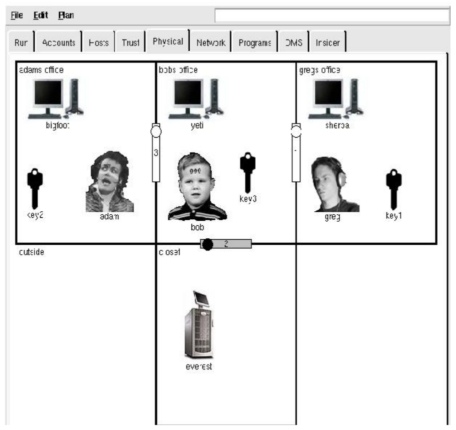
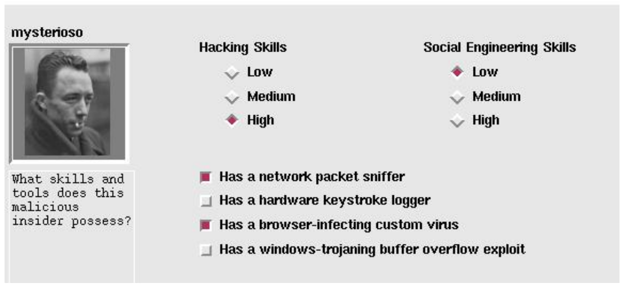


Figure 2: Top level screen of BAMS console.

The initial state of our office has Bob in a position to exploit several vulnerabilities. Bob has a key to his coworker's office. He has the ability to run a packet sniffer and also has a


mysterioso

What skills and tools does this malicious insider possess?

Hacking Skills: Low, Medium, [✓ High]

Social Engineering Skills: [✓ Low], Medium, High

- [✓] Has a network packet sniffer
- [ ] Has a hardware keystroke logger
- [✓] Has a browser-infecting custom virus
- [ ] Has a windows-trojaning buffer overflow exploit

Figure 3: Part of a dialog sequence for specifying a new malicious insider.

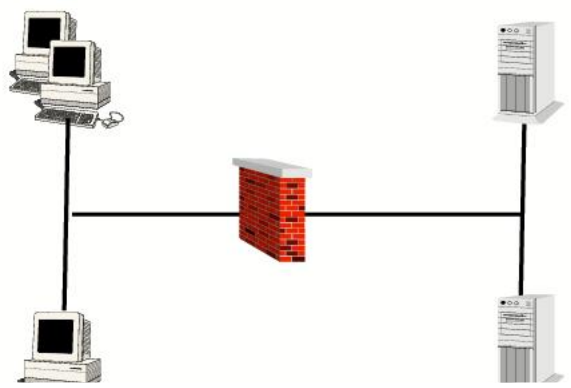


Figure 4: Network Configuration.

hacking tool which exploits a Windows^{TM} vulnerability and runs arbitrary code. Bob has two viruses, one that infects Internet Explorer^{TM} and forwards the data it receives and a second that runs in the background and forwards the whole screen. He controls both of these with the same program.

The layout of our network is as shown in Figure 4. The documents are served by a web-server on the organization's back office network to users on a front office network. The networks are separated by a packet filtering firewall that can be configured to control the connections of users to the servers. The front office environment is on a hub, which connects Bob's and his coworkers' computers. His coworker Greg is not the most security minded individual, and he is also somewhat gullible. Finally, while documents being transfered to or from the DMS are encrypted over SSL, administration traffic to the server is done over plaintext HTTP. Users authenticate to the DMS and the DMS admin server with fixed passwords.

The DMS has a simplified version of access control based on user or group permission to read or write a document. While the permissions are not as rich as a true ACL system allows, the end effects of who is allowed to read and write to a file are modeled. For purposes not involving subtle misconfiguration issues, this is sufficient to determine vulnerabilities.

This domain model was constructed in collaboration with and reviewed by experts in the U.S. intelligence community.

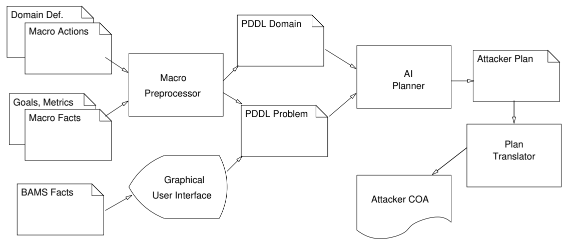


Figure 5: Top level view of the BAMS architecture.

While it is somewhat smaller than a real network to which such a tool would be applied, it possesses the appropriate characteristics. In particular, the authentication mechanisms implemented closely follow the "Community of Interest" model for cyber and information security.

BAMS Implementation

The BAMS system architecture used for our experiments is shown in Figure 5. The PDDL domain and problem specifications are provided as input files to the planner, which generates a plan.

As discussed in the next section, the PDDL versions of the domain model and planning problem are assembled from a set of modules written with the aid of custom macros designed to make writing large scale PDDL programs more natural and efficient. The Unix `make` utility invokes the M4 macro preprocessor (Seindal) to expand and assemble the sources as needed for a given problem. It also invokes a Perl program that processes the output of the planner and translates the plan steps to a more readable form. An example of the output of this post-processor is shown in Figure 6.

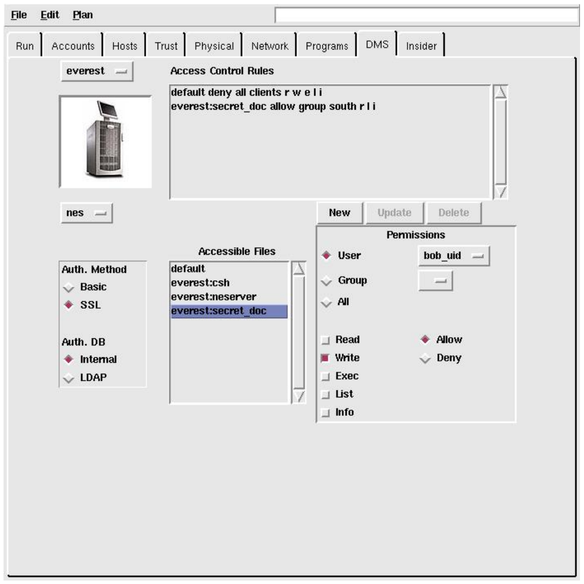
BAMS provides a flexible graphical user interface. The BAMS "console" can, within limits, configure the problem specification without the need for the user to write anything directly in PDDL. It consists of a set of tabbed dialogs, such as the one in Figure 7. In use of the system, the user selects the elements and options to present to the planner, and then issues a "RUN" command from the menu. The correct PDDL is generated, the planner invoked, and the output translated and displayed in the interface. Currently the console is used only to manipulate the fact base, not the domain definition.

The planner used in this system is Jörg Hoffmann's Metric-FF planner (Hoffmann 2003). METRIC-FF is a forward heuristic planner using a relaxed plan graph (Blum & Furst 1995) to provide heuristic estimates of the remaining distance from a state to the goal. These heuristic estimates are then used to guide search. With METRIC-FF there are two different search modes, Enhanced 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.

Before adopting METRIC-FF, we experimented with sev-

```
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 6: An automatically generated attack plan.



everest

Access Control Rules

default deny all clients r w e l i
everest:secret_doc allow group south r l i

nes

Accessible Files

default
everest:csh
everest:neserver
everest:secret_doc

Auth. Method
● Basic
[✓ SSL]

Auth. DB
[✓ Internal]
LDAP

Permissions

[✓ User] bob_uid
Group
All

Read
[✓ Write]
Exec
List
Info

[✓ Allow]
Deny

Figure 7: User interface showing a tab for configuring a Netscape based DMS on the host "Everest".

eral planners, choosing METRIC-FF for several reasons. First, it is fast enough to handle problems of a useful size and complexity. Second, METRIC-FF is thorough in its parsing of PDDL. All of the features we need are supported, including metrics and the use of complex logical expressions in the preconditions and effects of actions. Memory usage of METRIC-FF on our problems was reasonable, 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 memory if at all possible for the sake of speed. The scaling behavior of the planner on this domain is discussed below, in the section on performance.

Modeling Issues

We encountered several modeling challenges in this work.

Level of Detail

The first issue, evident from the beginning, is choosing the best level of abstraction at which to model the target system of interest. At too abstract a level, the resulting plans tend to be uninteresting, for example two-step plans such as: "the attacker gains root privilege on the server and then reads the secret document." At too detailed a level, (e.g. network packets or individual system calls) the plans are full of uninteresting detail and are far more difficult to generate. In addition, very detailed models are themselves labor-intensive to construct and maintain, and have a very short half-life.

Even minor changes to a network will require extensive rework.

Based on feedback from domain experts who have been involved throughout the project, we implemented the BAMS domain model at an intermediate level of detail. Actions encode adversary moves and exploits at a sufficient level of detail to model the possibility of a vulnerability, but not to make a rigorous determination, such as would be obtained through a rigorous parsing of firewall rules. Plans generated in our simplified domain with this model range from a dozen to a few dozen steps.

Review by domain experts has validated that the level of detail is appropriate, but indicate that the domain is still too small and the plans too short to be considered to be addressing a "real-world" domain. Informal estimates of the necessary increase in scale indicate a factor of about 10 required in the size of the domain model, and of about 2 in the length of plans.

A Natural Representation for Users

The second issue relates to usability. PDDL is a species of formal logic. Viewed from a different angle, PDDL is also a programming language. Neither of these representations are naturally comprehensible to the general user. In order for this system to be of any service to general users, the representation must be made transparent enough to allow them to describe the problem of interest with confidence. To avoid long periods of development, the representation should be lend itself to reuse.

We have added two enhancements to isolate users of BAMS from the need to directly represent facts and actions in PDDL. The first is a set of M4 macros that encapsulate some of the language's syntactic conventions and augment them 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. The full process of composing and sending an email message is shown in Figure 8.

An example of a macro used to define one piece of that process is shown in Figure 9. The first two "arguments" to the `defmodal` declaration are the modes, one required as a precondition for the PDDL action compiled out of this declaration, the second asserted as an effect. The presence of the same mode as both precondition and effect means that this action can be repeated an arbitrary number of times as part of the process, as shown in Figure 8. Having different initial and final modes permits a domain modeler to enforce the sequential occurrence of actions, with no other actions intervening.

The final problem and action definitions presented to the planner are assembled dynamically by M4 and a make script from component modules. This modularity allows us to partition all the predicates and classes pertaining to a single domain element (e.g. email, or doors and locks) into separate files, which are then merged and aggregated into full domain and problem specifications.

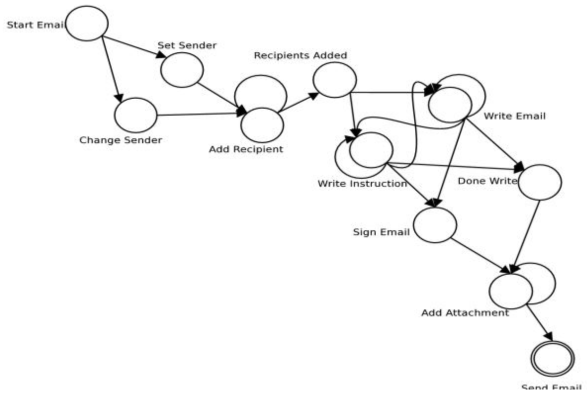


Figure 8: Composing and sending an email message

```
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 9: Definition for action ADD_RECIP, which adds a recipient to an email.

Modeling for Performance

For reasons of performance, we found it necessary to break large actions into smaller ones, which are then required to execute sequentially using the "defmodal" construct defined above. Consider the action `relay_viewed_doc` in Figure 10, which has a total of nine parameters, but only a single effect. A propositional planner such as METRIC-FF plans with ground instances of such an action, generated by instantiating the action as actions with the parameters replaced with all domain objects of the appropriate type. The number of grouund actions is then the product of the possible instantiations for each parameter, which for domains of even moderate size will be a very large number. For `relay_viewed_doc`, if we assume 3 possible instantiations for each parameter (so, 3 hosts on which the adversary may be logged in, for example), there will be $3^9 = 19,683$ ground actions to consider. This creates problems both in the initial generation of the propositional model, and potentially in searching for a plan as well, since any of these actions may be considered for addition to the current partial plan. In the domain model used for the experiments described below, a single action was segmented in this manner into as many as six smaller actions.

Deciding how to split actions to reduce the number of pa-

```
defaction(relay_viewed_doc
 :parameters (?doc - c_file
              ?human - c_human
              ?src_host - c_host
              ?s_proc - c_process
              ?src_proc - c_process
              ?malware - c_program
              ?dst_host - c_host
              ?dst_proc - c_process
              ?master - c_program)
 :precondition
  (and
   (at_host ?human ?dst_host)
   (viewing_doc ?src_host ?s_proc ?doc)
   (running_prog ?src_host ?src_proc ?malware)
   (can_transmit_documents ?malware)
   (running_prog ?dst_host ?dst_proc ?master)
   (can_receive_documents ?master)
   (net-connected ?src_host ?dst_host)
  )
  :effect
   (and (viewing_doc ?dst_host ?dst_proc ?doc)))
```

Figure 10: A large, complicated action

```
defmodal(free, r1,
  RELAY_VIEWED_DOC_1
 :parameters (?doc - c_file
              ?src_host - c_host
              ?s_proc - c_process
              ?src_proc - c_process
              ?malware - c_program
             )
 :precondition
   (and
    (viewing_doc ?src_host ?s_proc ?doc)
    (running_prog ?src_host ?src_proc ?malware)
    (can_transmit_documents ?malware)
    )
 :effect
  (and (transmitting ?src_host ?doc))
 )

defmodal(r1,free,
  RELAY_VIEWED_DOC_2
 :parameters (?doc - c_file
              ?human - c_human
              ?src_host - c_host
              ?dst_host - c_host
              ?dst_proc - c_process
              ?master - c_program)
 :precondition
  (and
   (transmitting ?src_host ?doc)
   (at_host ?human ?dst_host)
   (running_prog ?dst_host ?dst_proc ?master)
   (can_receive_documents ?master)
   (net-connected ?src_host ?dst_host)
  )
 :effect
 (and
  (viewing_doc ?dst_host ?dst_proc ?doc)))
```

Figure 11: Two smaller actions

rameters in each is more a matter of art than algorithm, but there are some heuristics. Effective splitting is dependent on how the parameters in the action's preconditions and effects can be separated. Ideally, parameters appear in a minimum number of sub-actions and each sub-action has a minimum number of parameters. As a final complication, those parameters with a minimum number of possible instantiations in the current domain model will ideally appear early in the sequence of sub-actions resulting from this split, due to an effect on search efficiency analogous to join query optimization for databases. Figure 11 shows the result of splitting the action in Figure 10. Note that we have had to add a new proposition `(transmitting ?src_host ?doc)` to enforce the same parameter instantiation in the two sub-actions.

Results

Scenario

For the results reported here, we used the domain described above, defining a scenario involving the successive detection and removal of vulnerabilities permitting successful attacks. The domain model includes 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 and somewhat gullible coworkers.

We selected a profile for our malicious insider "Bob" and gave him the goal of gaining unauthorized access to a secret document by exploiting a combination of physical, social engineering, and cyber exploits against a simple but realistic web-based DMS. After the planner generated a COA, we would identify and apply a counter-measure that rendered that specific attack infeasible. Application of these counter-measures was accomplished with minor, one and two line changes to the model. The scenario proceeded through seven attacks before sufficient safeguards were in place to prevent Bob from extracting the secret information. At that point, we added Adam the system administrator to the list of malicious insiders. This resulted in another pair of attacks that are only possible with Adam's collaboration.

Table 1 shows the plan lengths (in steps) and time to generate each plan in this process. The titles are interpretations that we provided to the synthesized plans. The final plan found required significantly more time than the others not just because it was longer, but because Enhanced Hill Climbing failed to find a plan, and METRIC-FF fell back to backtracking search. As discussed below, the search time was in general a minor component of the overall time required to find a solution, especially when EHC was successful.

The first COA that the planner generates is a stealthy attack that shows the ability of our tool to discover moderately complex plans of attack. The plan involved Bob starting his packet sniffer and waiting. Eventually Adam needs to do something to the DMS configuration and logs onto the server as the administrator. Since the environment is not switched, Bob is able to steal the administration password with the

Description	Length	Time (sec.)
Direct Client Hack	25	0.67
Misdirected Email	32	0.67
Shoulder Surfing	18	0.69
Email Trojan	37	0.71
Spoofed Email Trojan	37	0.73
Spoofed Instructions	36	0.79
Administrator ACL Change	23	1.20
Sniff Administrator Password	28	1.62
Sniff Password from Email	44	4.77

Table 1: Attack plan lengths and computation times

Min. Objects	Total Objects	Time Reported	Actual Time	Actions Generated
1	83	0.61	0.64	7433
2	89	0.7	0.73	7765
3	101	0.9	0.93	8598
4	116	1.22	1.28	11720
5	134	1.97	2.13	22102
6	153	3.71	3.83	38436
7	172	6.55	6.73	62074
8	191	14.44	15.37	94780
9	211	19.62	20.79	140205
10	231	33.84	157.13	199932

Table 2: Attack plan lengths and computation times

sniffer. Bob then uses that information to login to the server and change the ACL for the document he is interested in to allow himself read permissions. Following that operation he is able to download the document and gain access to the information held within.

To prevent this, we modify the firewall so it does not allow access to the DMS administration port's traffic on the firewall. Then we run the tool again to generate the second plan, a physical exploit where Bob wanders into Greg's office and reads data off the screen over Greg's shoulder. Note that this attack is qualitatively much different from the previous attack. In this instance the plan is shorter and much more risky. It also highlights a new potential attack method. To prevent this attack, we remove Bob's access to Greg's office. When we run the planner a third time, we finally see Bob launching a direct hack against Greg's machine. The attack launches one of Bob's viruses that stays resident and forwards the screen buffer to Bob's machine. When Greg loads the document that Bob is interested in, Bob is able to read it as well. This process continues until no plan is found.

Performance

In any application of a propositional planner to a problem of even moderate size, memory management is both difficult and crucially important. For example, the memory required for the ground representation of an action grows as the product of the domain sizes for each parameter. So, memory usage grows roughly as $m * N^X$, where m is the number of actions, N is the number of objects in the domain and X is the number of parameters in an action.

However, there are some simple optimizations that can help quite a lot. As discussed previously, one thing we can do is to reduce the number of parameters per action, by splitting them into sequences of smaller ones. Another simple optimization that had not been made at least in the version of METRIC-FF that we downloaded was to use type information in allocating space for ground actions. With these two simple optimizations, memory usage grows as $m * n^{X/a}$ where n is the number of objects *of a given type* in the domain, and a is the fraction by which the number of parameters per action has been reduced.

With specific reference to METRIC-FF, at least to the version of the tool that we have been using, we found other opportunities to reduce memory consumption through simple optimizations, for example by reducing the size of number fields to the minimum required. In reachability analysis, some data fields that were 32 bits are now 8 bits in our implementation, and may be shrinkable to 1 bit. Another simple, though more laborious, optimization involves rewriting actions to avoid what in METRIC-FF are called "hard action templates," which appear to be actions whose preconditions METRIC-FF cannot figure out how to reduce to DNF. For example, rewriting a precondition `(and (foo) (or (bar) (baz)))` as `(or (and (foo) (bar)) (and (foo) (baz)))` does not change the semantics, however the former results in a hard action, the latter does not.

In the current implementation, search is not a primary consumer of time or memory. According to the statistics reported by METRIC-FF, the total run time when the whole problem fits in memory (so, when swapping to disk is not a factor) is dominated by time spent constructing hard action templates. This is testimony to the efficacy of the Enhanced Hill Climbing (EHC) heuristic, which finds a solution in most of our problem instances. This is not entirely by chance: our domain has been constructed and empirically modified to facilitate the performance of EHC, which is not always easy, and does not always result in a natural representation of the problem.

Table 2 provides a more quantitative analysis of the scaling behavior of the resulting system. To test how time and memory usage scale with domain size, we added additional objects to an initial problem domain, then timed how long it took to load the domain description, perform the initial generation of the propositional model on which METRIC-FF actually plans, and find a plan. The domain contained more than 20 different object types. Increasing the minimum number of each type of object yields reasonable growth in time through a minimum of 7 of each type of object. The system manages to return a result in fewer than seven seconds for problems with 172 objects, when there are no fewer than 7 of any single object present.

After at least 8 of every object are present in the problem the exponential growth of action templates begins to tell. Much of the total time is spent constructing these action templates. With 10 of each object, limitations on our test machine's memory were exceeded and METRIC-FF began to swap, as can be seen in the discrepancy between reported and actual elapsed time. On the other hand, the worst case reported is less than three minutes for a problem with 231 separate objects and more than 50 actions, combining to generate nearly 200,000 ground actions.

Informal estimates of the necessary increase in scale indicate a factor of about 10 required in the size of the domain model, with much of that increase taken up in the "breadth" of the model (i.e., new actions, propositions, and object types), and a factor of 2-5 in the length of plans. Given the results reported above both of these increases appear to be feasible, but have not yet been accomplished.

Related Work

Alternative Planning Appraches

METRIC-FF is one of a broad family of forward heuristic planners that could have been applied to this problem. We chose it over others for several reasons, the most prominent being the range of pddl that METRIC-FF will parse correctly, the performance of the planning algorithm itself, and the fact that Hoffmann very kindly makes the source available, permitting us to make some minor but necessary modifications. We were philosophically attracted to backward planners, in particular given our interest in generating multiple plans, but could not find any that were sufficiently expressive and would scale to the size of the problems needed. Finally, HTN planning is a strategy we rejected, specifically because we were interested in novel combinations of actions to form new attacks, which renders irrelevant some of the main strengths of a hierarchical approach.

Other Approaches to Vulnerability Analysis

Other researchers have used automated methods to perform various sorts of vulnerability analysis. In (Wool 2001) 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 (Zerkle & Levitt 1996) 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 (Phillips & Swiler 1998). 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.

Ritchey and Ammann propose a model checking approach (Ritchey & Ammann 2000). Using the BDD-based model-checker NuSMV, 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 (Sheyner *et al.* 2002), developing techniques for identifying a minimal set of security countermeasures to guarantee a safety property, and to compute the probability of attacker success associated with each attack graph.

While we have not performed head-to-head comparison experiments, the performance numbers reported for these approaches lag the result reported here by orders of magnitude. 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. In more recent work, Sheyner uses an explicit state model-checking approach which appears to scale much better, but still grows linearly with the size of the entire state space (Sheyner 2004).

The work on "Network Hardening" by Noel et al. (Noel *et al.* 2003) is very close to ours in approach. Their "exploit graph" is quite similar to, though apparently developed independently from, work on plan graphs.[1] Their assumption of monotonicity in an attacker's access to the system corresponds to one of the simpler of a large family of "tractable subclasses" of AI planning (for example, (Jonsson & Backstrom 1998)). This is not an assumption we have made, and it is not yet clear whether the assumption is warranted in the more expressive cyber/social/physical domain model that we have developed

Future Work

There are a number of issues raised in this application of classical planning techniques to vulnerability analysis where further work would be beneficial.

Systematic Exploration of the Domain

The work reported here involved experiments with a single domain model, tweaked fairly extensively by hand. We did not get to the point of writing a domain generator, which would have allowed us to produce a large set of problem instances varying in systematic ways.

Model Extension

PDDL is essentially a programming language. The use of PDDL or other planning formalisms in complex domains can involve or mimic many of the same structures required in programming, for example variable binding, method specialization, and even something like the Lisp `gensym` function. As in any such language, construction of a correct and efficient "program" generally requires considerable expertise.

The general issues that must be addressed in extending the model include:

- Recognizing when new predicates are required to capture state, and defining ones that do so efficiently.
- Knowing how to scope and parameterize operations that capture attacker capabilities.

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

- Managing interactions between new operations and previously defined ones.

Action interactions are perhaps the trickiest problem, due to the difficulty of isolating different aspects of the model. For example, if we wish to model the fact that an attacker can recover secret information from the swap space of the disk, a new "swap-snooping" operation would be added. But to be complete, we must also model all of the existing operations that could leave readable information in the swap space. For example, if an email program caches a password, then this datum might find its way into the swap area.

One approach is to use a domain theory to automatically infer some ramifications of actions. Instead of modeling everything directly in PDDL, some of the model is declared as axioms in a domain theory, which is then used to make the necessary inferences to build complete action definitions. We have demonstrated (in a preliminary way) that you can indeed simplify planning models using such as the action language of Lin (Lin 2003).[2]

An example fragment of such a theory is shown in Figure 12. In the example, the "causes" clauses state some very obvious axioms about objects with physical extent. The action definition specifies the action parameters, preconditions and effects as usual, but the effects part is very simple–it suffices to state here that the person is carrying the key. The side effect of the key no longer being available in the room is added automatically by the domain theory when the action definition is expanded, as shown in Figure 13. The Prolog program that reasons about the domain to produce the actions is based on the one created by Lin, but substantially modified by us to generate the full range of legal PDDL we need. Note that it also removes variables by substituting all legal values in the domain, thus precompiling ground actions. This is an artifact of the algorithm that Lin uses to compute context-dependent effects and other ramifications of actions. The example shown here is simplified for illustrative purposes, and could also have been handled using the "derived predicates" (re)introduced in PDDL 2.2

```
% . . .
% if H is carrying K, K is not in a room

causes( carrying(H,K), -in_room(K,R) ) :-
   fluent(carrying(H,K)),
   fluent(in_room(K,R)).

% vice versa

causes( in_room(K,R), -carrying(H,K) ) :-
   fluent(carrying(H,K)),
   fluent(in_room(K,R)).

% nothing can be in two rooms at once!

causes( in_room(H,R1), -in_room(H,R2) ) :-
   fluent(in_room(H,R1)),
   fluent(in_room(H,R2)),
   R1\==R2.

%  person h picks up the key k in room r,
%  only if h and k are both in r.

action(grab_key(H,K,R)) :-
   c_human(H), c_key(K),c_room(R).
poss(grab_key(H,K,R),
     in_room(H,R) & in_room(K,R)).
effect(grab_key(H,K,R),true,carrying(H,K)).
```

Figure 12: Fragment of domain theory in an action language prototoype for BAMS. Syntax is Prolog. The hyphen indicates negation.

```
%% automatically generated PDDL action

(:action grab_key_bob_key1_room2
 :precondition
  (and (in_room bob room2)
       (in_room key1 room2))
 :effect
 (and
   (carrying bob key1)
   (not (in_room key1 room2))))
```

Figure 13: PDDL action generated from the action language definition in the example domain theory.

Bottleneck Analysis

Currently, BAMS functions as an interactive support tool for a security analyst who interprets the plans. As an additional level of automation, we would like to add a capability to analyze a set of COAs for a given domain and set of attackers, so as to identify countermeasures that would defeat the most attacks, or the most threatening attacks. One way to accomplish this would be to generate multiple plans to achieve a given attacker goal, thus enabling comparative analysis.

One of the unaddressed difficulties is generating multiple plans that are different in *interesting* ways. For example, two plans that differ only in the process identifier attached to a running program are fundamentally the same. In addition, there are efficiency considerations. METRIC-FF and other forward heuristic planners extend plans to adding actions to a prefix. There is no link between the added action's effects and the eventual goal, though the hope is that an effective distance estimate will strongly favor the addition only of relevant actions. For the generation of a single plan, our experience is that this usually works well. For generating multiple plans, there is a problem in situations involving large domain models and alternative plans of significantly different lengths. Suppose that the forward planner finds one plan, of length n. Suppose that there is a second plan that will also achieve the goal, of length $n + k$. For k even slightly larger than 1, a forward planner naively invoked to keep generating additional plans will generate the plan of length n, then will proceed to generate a large number of plans of length $n + 1$ by inserting all actions that are irrelevant but not interfering,

[2]For a variety of reasons having to do with efficiency, customization, and limited resources, we currently employ the macro facility m4 to do this compilation.

at all points in the plan where they are both enabled and not interfering. Similarly for plans of length $n + 2$, using pairs of non-interfering actions. The combinatorial problem, especially for large domains, is obvious.

An alternative method of generating multiple plans from the same problem would be to use a system explicitly constructed for systematically generating multiple plans, such as Zimmerman's MULTI-PEGG (Zimmerman & Kambhampati 2002), though there are still some questions as to whether these approaches, and MULTI-PEGG in particular, will scale to address problems of realistic size. Other possible ways to speed up this process include the use of algorithms for tractable subclasses of planning models (Jonsson & Backstrom 1998), or eschewing the explicit generation of plans altogether, instead extracting information directly from the plan graph, for example using *landmarks* as described in (Porteous, Sebastia, & Hoffmann 2001).

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. And for the defender, some attacks will carry a higher cost than others. A natural extension for future work would be to incorporate this information into the process of COA generation and analysis (and thus quite likely into the planning process).

Summary and Conclusions

In this work, we have demonstrated the application of classical AI planning in a novel and interesting domain. 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. We have applied our implemented system to a domain involving insider attacks on a simple, yet realistic web-based document management system, resulting in the generation of reasonably complex attacks consisting of twenty to fifty steps in a few seconds.

This work raises some interesting issues for planning as a field, as well. The planning performance required in BAMS pushes the state of the art in several directions. Domain model engineering and maintenance, matching planning algorithms to problem characteristics, optimizing planner performance, and the significant interactions among these issues, are all areas that we had to address.

There is ample future work to be done in a number of different areas. To that end, one of our near-term objectives is the generation of a version of the BAMS planning problem that can be freely shared with other researchers. This material will be available on the Adventium Labs' website (http://www.adventiumlabs.org) prior to this paper's publication in early June.

Acknowledgements

The authors would like to thank three anonmymous reviewers, for providing unusually detailed and constructive comments on the submitted version of this paper.

References

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Fox, M., and Long, D. 2002. Pddl2.1: An extension to pddl for expressing temporal planning domains. Technical report, University of Durham.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*. accepted for the special issue on the 3rd International Planning Competition.

Jonsson, P., and Backstrom, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1-2):125–176.

Lin, F. 2003. Compiling causal theories to successor state axioms and strips-like systems. *Journal of Artificial Intelligence Research* 19:279–314.

Noel, S.; Jajodia, S.; O'Berry, B.; and Jacobs, M. 2003. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of 19th Annual Computer Security Applications Conference*, 86–95. IEEE Computer Society.

Phillips, C., and Swiler, L. P. 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the New Security Paradigms Workshop*, 71–79.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 01)*.

Ritchey, R. W., and Ammann, P. 2000. Using model checking to analyze network vulnerabilities. In *Proceedings 2000 IEEE Computer Society Symposium on Security and Privacy*, 156–165.

Seindal, R. *GNU m4*. GNU Software Foundation. http://www.seindal.dk/rene/gnu/.

Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; and Wing, J. M. 2002. Automated generation and analysis of attack graphs. In *2002 IEEE Symposium on Security and Privacy (SSP '02)*, 273–284. Washington - Brussels - Tokyo: IEEE.

Sheyner, O. 2004. *Scenario Graphs and Attack Graphs*. Ph.D. Dissertation, Computer Science Department, Pittsburgh, PA.

Wool, A. 2001. Architecting the lumeta firewall analyzer. In *Proceedings of the 10th USENIX Security Symposium*. Washington, D.C.: USENIX.

Zerkle, D., and Levitt, K. 1996. NetKuang–A multi-host configuration vulnerability checker. In *Proc. of the 6th USENIX Security Symposium*, 195–201.

Zimmerman, T., and Kambhampati, S. 2002. Generating parallel plans satisfying multiple criteria in anytime fashion. In *AIPS-02 Workshop on Planning and Scheduling with Multiple Criteria*.