Planning for Security Management

Rosy BARRUFFI and **Michela MILANO** and **Rebecca MONTANARI**
DEIS
University of Bologna, ITALY
{rbarruffi,mmilano,rmontanari}@deis.unibo.it

Abstract

Security Management is a key issue in distributed computer systems. Resources and data need to be protected against unauthorized access, manipulation and malicious intrusions that render a system unreliable or unusable. The complexity of the task calls for the design of intelligent support systems that aid system administrators in the detection and/or prevention of intrusions. For this purpose, Intrusion Detection Systems (IDS) have been deeply investigated. IDSs are aimed at identifying intrusions and triggering consequent repair and/or reconfiguration actions. In general, these recovery procedures are statically defined by a system administrator. An alternative approach relies on a planner that dynamically computes the action chain (plan) for reconfiguring/repairing an attacked system. Using planning techniques greatly increases IDS flexibility, since statically defined countermeasures are not always the most appropriate and can be excessive (or even wrong) in some situations. In this paper, we discuss the design and implementation of a constraint-based planner that acts as a reacting module in an IDS.

Introduction

Security Management is beginning to assume enormous importance in today's computing environment. Organisations are increasingly embracing the potential of the Internet as a powerful, low-cost medium for business transactions that include product marketing, advertising, electronic trade and customer support. Even though the exploitation of network services grants unarguable advantages, it also greatly increases the risk of security breaches. As a consequence, it is very important that security mechanisms are designed in order to protect data from disclosure or unauthorized manipulation, and to prevent denial of services.

In distributed computer systems, the complexity of security management calls for the design of automated systems for the detection and/or prevention of intrusions. However, completely automating security management appears, at present, unrealistic. We can, however, develop intelligent support systems that assist network administrators in managing security. For this purpose, in the last years, Intrusion Detection Systems (IDS) (Lunt 1993b) have been deeply investigated. IDS are in charge of identifying intrusions and triggering consequent repair and/or reconfiguration actions that lead the system from a faulty state to the correct one. IDS repair and reconfiguration mechanisms usually rely on complex procedures (to be written by the system administrator) that take into account each situation that is likely to happen.

This approach represents an efficient solution, but might not be feasible for complex and dynamical environments where actions interact and can be combined in many different (sometimes unpredictable) ways, thus leading to long development time and resulting in potential software bugs. In addition, statically defined procedures might not be always the most appropriate countermeasures and can be excessive (or even wrong) in some situations. This happens, for instance, when IDS false alarms are triggered or when system security policies dynamically change, leading to adopt different countermeasures for the same attack situation. Thus, flexibility becomes an essential requirement for security management.

We argue that this feature can be achieved by the adoption of planning techniques for dynamically computing repair/reconfiguration plans. Plans are sequences of elementary actions built according to the intrusion occurred, the current system state and security policies enforced. We propose the design of an IDS reactive component relying on a constraint based planner able to produce recovery/reconfiguration plans given a potential intrusion. During the plan construction, the planner is able to interact with the system in order to retrieve relevant information on the current resource and process state.

Intrusion Detection Systems and Planning

IDSs

Intrusion detection technology is aimed at identifying intrusions against computer systems. An intrusion can be defined as any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource. Auditing is used to determine whether and how a security violation has occurred; it consists of the examination of the history of system activities that is recorded to a file, called audit trail, in chronologically sorted order. However, audit trail analysis is highly complex due to the huge amount of data they contain, typically in a raw and difficult format to understand. Support to automation is, thus, required. The so-called Intrusion Detection Systems are tools aimed at automating both the audit data acquisition and audit trail analysis.

Two main types of Intrusion Detection techniques have been developed:

- anomaly detection (Teng, Chen, & Lu 1990; Lunt 1993b) aimed at modeling correct and acceptable user behaviour and resource utilization: intrusions are, then, detected when user or system behaviour differs from the correct one. It is not required a priori knowledge of possible security flaw from which the system may suffer.
- misuse detection (Kumar & Spafford 1995; Garvey & Lunt 1991; Lunt 1993a) aimed at modeling intrusions as patterns: intrusions are detected via pattern matching with the model. Only known system vulnerabilities and attack scenarios are identified.

No single approach can be considered satisfactory for all types of intrusions. Each approach is appropriate for detecting a specific subset of violations.

A general architecture for IDSs, called the Common Intrusion Detection Framework (CIDF) is defined in (Kahn *et al.* 1998). CIDF comprises an event-generator unit (E-box), an analysis engine (A-box), a storage mechanism (D-box) and a response module (R-box) (see Figure 1). The E-box monitors the environment in order to provide information on the system state to the other IDS components. The A-box analyzes the monitored events provided by the E-box and audit trails in order to detect suspicious or malicious activity. In particular, the A-box adopts one of the intrusion detection techniques previously described. The D-box stores security information in order to make it available to system administrators at a later time. The R-box reacts to detected intrusions either by preventing (pro-active behaviour) or recovering (reactive behaviour) from them. The pro-active or reactive behaviour depends in general on the type and the accuracy of the information provided by the A-box.

The set of reactive actions is generally statically defined (and should be encoded in the IDS by the system administrator) on the basis of the intrusion detected by the A-box. This might not be a flexible solution as it does not allow to automatically deal with changing and unpredictable situations. Moreover,it requires from system administrators a deep prior knowledge of potential intrusion scenarios.

We, thus, propose an alternative approach which offers a more flexible solution to countermeasure definition by dynamically building repair plans. Our R-Box relies on a planner able to compute appropriate reconfiguration/recovery plans on the basis of both the system state and the final state (goal) to achieve.

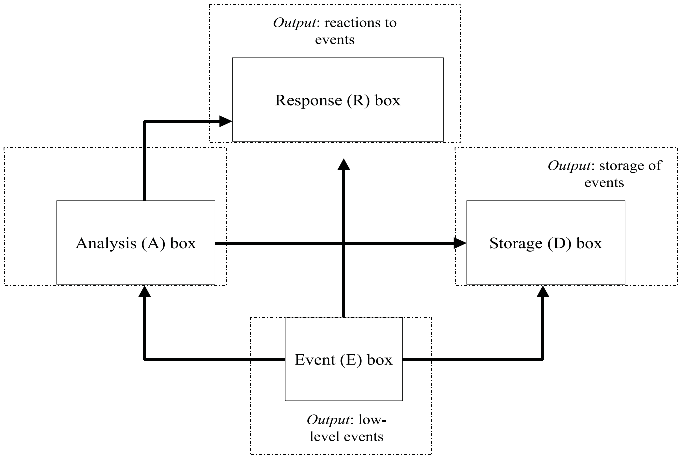

Figure 1: CIDF components

Planning Techniques

A Planner is an Intelligent agent which dynamically synthesises the sequence of actions (plan) necessary to achieve a desired state (goal) starting from a given initial state of the system. The basic actions used by the planner are defined by means of preconditions, which represent the conditions that need to be satisfied in the current state in order to execute that action, and post-conditions, which represent the effects the action causes on the world, so as to change its current state. Note that modern planners work with action schemata with variables (Weld 1994) defining classes of actions more than with single, completely instantiated actions.

We consider a regressive non linear Partial Order Planner (POP) (Weld 1994) able to create partial plans corresponding to the achievement of different subgoals and at the same time to cope with the resolution of threats due to the interference among subgoals. POPs in general perform least commitment planning, that means that decisions are made only if and when constraints force to do it. In particular, only the essential ordering decisions are recorded so that plans are represented as a partially ordered sequence of actions avoiding to prematurely commit to a complete totally ordered plan. An efficient way of allowing partial order planners to postpone decisions consists in introducing constraints and actively maintaining their consistency during the refinement of the plan, as proposed in (Joslin & Pollack 1995).

The POP algorithm receives three inputs (Weld

1994): (*i*) the description of the world initial state; (*ii*) the required goal; (*iii*) the set of actions that can be performed on the system. POPs build the plan searching (backward, starting from the required goal) over partially specified sequences of actions (*partial plans*). All the conjuncts of the final goal are initially put into an *Agenda* representing the set of "open" condition that the planner needs to satisfy. The POP algorithm considers all the atomic formulas expressing the initial state as the post conditions of a dummy action of the plan called *Start*. At each step of the planning process, the planner either performs open condition achievement or threat resolution. In the first case, the planner selects an open condition Q from the *Agenda* and tries to satisfy it by searching for an action whose effects contain a conjunct unifying with Q. This action can be either a newly instantiated action or an action already in the plan. In particular, if *Start* is selected, it means that that condition is already satisfied in the initial state. On the other hand, when a new action is introduced into the plan its preconditions are added to *Agenda*.

Threat resolution is triggered when different actions introduced for different subgoals interfere with each other. A threat occur when post conditions of an action belonging to a partial order chain negate pre conditions of a second action in another chain. In order to solve threats the planner imposes appropriate ordering constraints among clashing actions. Constraint propagation techniques can be used in order to increase the efficiency of the planner by reducing the search space to be explored. Planners making active use of constraints are referred to as *constraint-based* planners, see for example (Joslin & Pollack 1995; Yang 1992; Yang & Chan 1994; Kambhampati 1996; Lever & Richards 1994; Tate, Drabble, & Dalton 1994). The idea is to actively maintain the consistency of constraints during the plan construction, as previously proposed in (Joslin & Pollack 1995; Kambhampati 1996; Yang 1992). Many previous work to the use of constraints in planning are known ((Joslin & Pollack 1995; Yang 1992; Yang & Chan 1994; Kambhampati 1996; Lever & Richards 1994; Tate, Drabble, & Dalton 1994)).

We have extended a traditional constraint-based POP in two ways: on one hand, we have used constraint satisfaction techniques in order to apply least commitment on the action variables binding activity, and not only for delaying ordering decisions.
On the other hand, while traditional planners assume that the world initial state is completely known at the beginning of the computation, in distributed systems, this assumption is unrealistic because of the enormous amount of data to be stored and continuously updated. Relevant works have been proposed for extending traditional planners in order to cope with incomplete and dynamic knowledge (Draper, Hanks, & Weld 1994; Golden 1997; Olawsky & Gini 1990; Golden & Weld 1996). Most of them gather information by means of declarative sensing actions inserted into the plan. In our system, we adopt an alternative approach where the knowledge acquisition activity is demanded to the constraint solver, thus being transparent to the planner.

The R-Box

In our approach, the IDS reactive module (R-box) is a goal-driven reacting component. While responding to the alarms reported by the A-box, our R-box dynamically pursues goals on the basis of the changes occurred in the computing environment. In particular, it receives as input a diagnosis of the detected attack coming from the A-box in form of a 5-tuple(ISS 1998): {`event`, `event_type`, $\mathtt{T}_{intrus}$, `source`, `dest`} where `event` represents the specific intrusion, `event_type` the intrusion category, $\mathtt{T}_{intrus}$ the time in which intrusion has occurred, `source`, `dest` the source and destination of the intrusion. As concern the term `event_type`, we cope with five different categories (Smaha 1998): (*i*) *attempted break-ins*, they aim to break-in a system by exploiting its vulnerability, e.g., attempts to gain access privileges; (*ii*) *penetration of the security control system*, i.e., successful attacks, consisting in obtaining unauthorized access to files, programs or computer system control; (*iii*) *leakage*, i.e., information acquisition from unauthorized recipients; (*iv*) *malicious use*, i.e., resource loss or manipulation; (*v*) *denial of service*, aimed at rendering a service unavailable.

The R-box relies on two components: a Goal Manager (GM) and a Planner. Given the event, the GM selects the appropriate recovery/reconfiguration goals according to system policies. Then, the Planner builds a recovery plan for achieving the goal by interacting with the underlying system for retrieving information during the computation.

The Goal Manager

The Goal Manager (GM) is in charge of generating the declarative definition of the final conjunction of goals to be satisfied in order to go back to a safe state of the system. GM internal structure (see Figure 2) consists of two components: (i) an *Event Refinement Module (ER)* and *Policy-based Goal Refinement Module (PGR)*.

The ER refines the intrusion classification specified by `event_type` on the basis of the end-effects produced by intrusions on system resources and processes. End-effects can be classified as: (*i*) unauthorized *resource* access; (*ii*) unauthorized *resource* manipulation or loss; (*iii*) unauthorized operations on *processes* including deletion, insertion, or exhaustion of critical resources (e.g., CPU, memory). Thus the output of ER is given in the form `event_effect(Effect, X)`. For example, `event_effect(not_process_safe, sendmail)` indicates unauthorized operations on the mail delivering service `sendmail`. The corresponding intermediate goal is, thus, `process_safe(sendmail)`. The ER output is refined by the PGR module on the basis of the system policies. Policies are guidelines expressing the

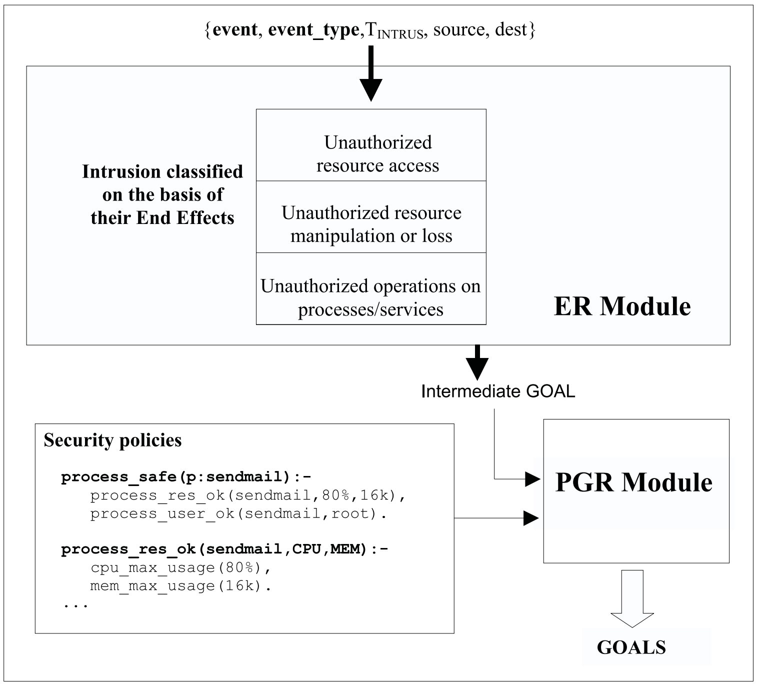

Figure 2: Goal Manager Structure

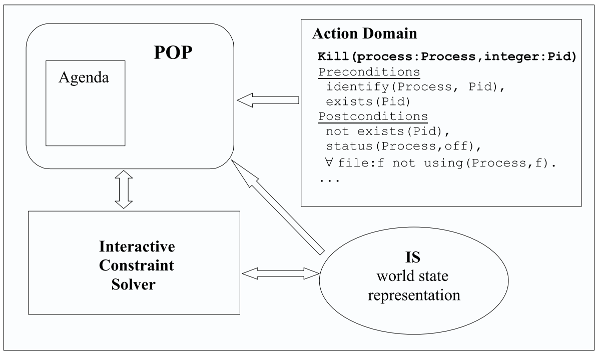

Figure 3: The planner

correct and expected behaviour of the system. In the architecture proposed, they are expressed as rules. An example is depicted in figure 2 where a rule in the Security policies block is reported stating that the resource utilisation of process `sendmail` in order to be not stalled is 80% for CPU and 16Kb for memory. In addition, the owner of the process `sendmail` should be `root`. An usual resolution process is started that produces the set of final goals. Each rule is eventually refined by considering each literal in the body and looking for another rule whose head matches with the literal. Those literals not matching with any rule are considered part of the final goal as residues. In figure 2, for example, the literal `process_res_ok(sendmail,80%,16k)` matches the head of the second rule. Thus, it is replaced in the final goal by the literals `cpu_max_usage(80%)` and `mem_max_usage(16k)`.

The Constraint Based Planning Agent

As already mentioned, the planner used to achieve the final goal coming from the R-Box GM component, is an extension of a constraint based POP which has been fully implemented in the Constraint Logic Programming language ECL^iPS^e (ECRC 1992). Constraint Logic Programming (Jaffar & Maher 1994) is a class of programming languages combining the advantages of Logic Programming and the efficiency of constraint solving. In figure 3 the planner architecture is depicted along with an example of action schemata devoted to killing a given process identified by a unique `Pid`. The preconditions which need to be true for executing the action `Kill` are that the identifier of `Process` is `Pid`, and that it exists in the system. The effects of action `Kill` are clearly that the process identifier does no longer exist, the status of the process is off and it does not use any file.

Note that more complex scripts can be designed as basic building blocks for building plans. Obviously, the more complex are basic actions, the shorter are resulting plans and thus more efficiently computed. Therefore, we need to find a tradeoff between complexity of basic actions (which should be written once for all by a system administrator) and the efficiency of the planner.

Let us see how the planning search algorithm works. As already mentioned above, we consider that the planner does not have a complete knowledge of the initial state thus an information gathering mechanism is needed. We have exploited constraint satisfaction techniques both for delaying action variables commitments and for dealing with knowledge acquisition. The planning problem is mapped into a Constraint Satisfaction Problem (CSP): a CSP is defined on a set of variables ranging on finite domains of values, and a set of constraints. The planning problem can be viewed as a CSP whose variables are those appearing in pre and post condition of action schemata introduced into the plan (Yang & Chan 1994). The constraints are represented by the usual nocodesignation ($\neq$) and codesignation ($=$) constraints imposed among the problem variables during the planning process. Variable domains are represented by sets of objects in the domain description and they are pruned thanks to constraint propagation in order to remove values which are not consistent with constraints. Constraint propagation allows the planning algorithm to avoid wrong instantiation and consequent computationally heavy backtracking steps.

Standard constraint based approaches need all the information on variable domains at the beginning of the computation. We have properly extended the CSP framework in order to treat constraints on variables ranging on partially or completely unknown domains. The propagation of those constraints, called *Interactive Constraints* (IC), may result in a knowledge acquisition process during the plan construction. For a formal definition of the Interactive CSP framework see (Barruffi *et al.* 1998; Barruffi & Milano 1998; R.Cucchiara *et al.* 1999).

In order to exploit this mechanism we consider action preconditions and final goals (i.e., open conditions) as Interactive Constraints. When open conditions are selected from *Agenda*, our planner first tests them in the initial state relying on a constraint solver. Therefore they are propagated as ICs: if they work on variables ranging on known domains, traditional propagation is performed, otherwise knowledge acquisition is performed in order to acquire domain values. It is worth noting that, only information consistent with constraints is retrieved, so as to simplify the task of pruning inconsistent alternatives. These constraints are awaked, during the planning process, when some variables get instantiated or their domain pruned, until the correspondent variables are instantiated. For instance, consider the action `Kill` presented in figure 3, the constraints are `identify(Process,Pid)` and `exists(Pid)`. `Process` and `Pid` are domain variables containing possible values, i.e., process names and identifiers respectively. When variable domains are unknown constraint propagation retrieve values, i.e., process identifiers consistent with constraints `identify(Process,Pid)` and `exists(Pid)`. These process identifiers represent the domain of `Pid`. This domain can be eventually pruned thanks to another constraint, say `owner(root, Pid)` which removes from `Pid` domain all identifiers whose owner is not `root`. When a variable domain becomes empty, the involved constraints fail meaning that they can not been satisfied in the initial state. The planner needs to perform a typical action search step in order to satisfy them.

An important point should be discussed concerning knowledge acquisition. In general, as suggested in (R.Cucchiara *et al.* 1999), when performing knowledge acquisition, we can acquire one consistent value or all consistent values with the interactive constraint. The corresponding search algorithm in the two cases have been called *interactive forward checking* propagation algorithm and *interactive minimal forward checking* technique. The former method results in an acquisition of all consistent values, while the latter consists in retrieving only one consistent value at each propagation step. The two methods performances depend on the domain of application. Usually "total forward checking" is a more suitable propagation technique in those applications where gathering a big deal of information once for all is less expensive than querying the system many times. The opposite reasoning values for *minimal forward checking*. In our case, in general, the first technique has been experimentally identified as the most effective.

The interaction with the real world allows the planner to deal with *false alarms* deriving from wrong A-box esteems. In the case of false alarms, it may happen that statically defined actions are triggered that, for example, block legitimate traffic or close valid connections. The IDS itself becomes a source of denial of service. In our architecture, given an alarm and the corresponding goal, the planner tests the current system state. As a consequence, false alarms would not trigger any undesirable countermeasure.

An Example of Attempted-break-in

In this section, we sketch an example (see figure 4) of reconfiguration/recovery plan in the case of an *attempted break-in* intrusion, called*smtp-wiz* attack. The intruder exploits a *backdoor* of the e-mail server program sendmail (`sm` in figure 4) deriving from an incorrect service configuration, gains access through the sendmail port and executes arbitrary commands on the system that may damage both resources and system processes.

Given the A-box output, the ER module produces intermediate goals corresponding to the "attempted break-ins" event_type. These goals are refined by the PGR module according to the security policies, until the set of final goals is produced. We focus on goals `G1`, `G2` and `G3` in figure 4 aimed at: (*i*) re-establishing the availability of the system processes specified in the security policies; (*ii*) killing all extra unauthorized system processes; (*iii*) correctly configuring the e-mail service (so as to prevent further attacks of the same type).

Thanks to interactive constraints, once `G1` is selected from the *Agenda*, the planner checks if each process included into the set of authorized system processes (`list` in figure 4) is running. Otherwise, action `Activate(process:P)` is selected for restarting `P` (`x1` in figure 4). For `G2`, the planner retrieves all running processes (i.e., the domain of variable `Y`) and prunes from the domain all values not belonging to `list`; for those left, it searches for an action targeted at killing them (`Kill(process:P,int:Pid)`). When `G3` is selected, the planner chooses the action (`Ch_conf(process:P,string:Rule)`) for properly configuring the e_mail service by inserting the `anti_smtp-wiz` rules in the sendmail configuration file. We assume that configuration files can be changed only when the corresponding processes are off. Thus, the planner places before `Ch_conf` a `Kill` action. Due to `G1` sendmail is restarted.

Related work

Security and, more generally, system management are attractive testbeds for AI technology. As far as security

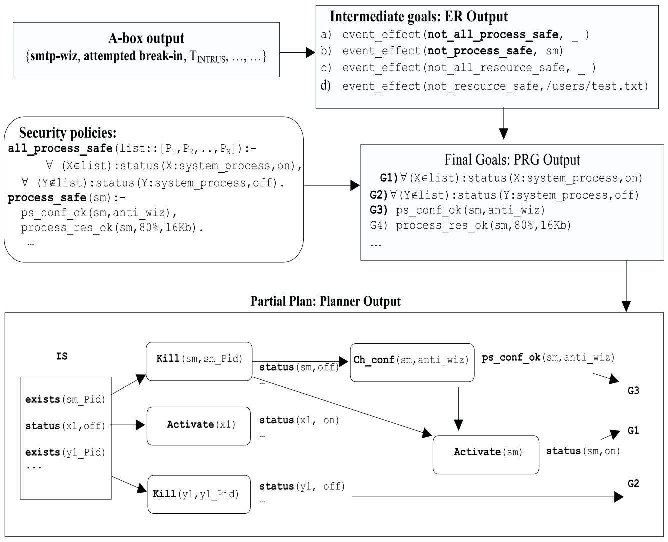

Figure 4: Plan example

is concerned, we are aware of few approaches based on planning techniques. An interesting project entitled *Explaining and Recovering from Computer Break-ins* carried out at the SRI AI Center has lead to the design of an integrated prototype system DERBI (Tyson *et al.* 1998) for diagnosing and recovering from intrusions. The prototype is based on a Procedural Reasoning System (PRS) (Myers 1996) that has been developed for using an expert's procedural knowledge for accomplishing goals and tasks. The PRS architecture contains three main modules: (*i*) a database containing current facts and beliefs (*ii*) a set of goals to be achieved and (*iii*)a set of plans describing how actions may be performed to achieve certain goals or react to certain situations. PRS represents a reactive, goal driven engine. Currently, only DERBI diagnosis functionalities have been developed. They will be extended with recovery capabilities so as to assist system administrators.

Further interesting work in the field of IDSs is (Garvey & Lunt 1991) where planning techniques are used for different purposes w.r.t. ours: in order to detect intrusions, an *anticipator* is used for predicting the next step in the scenario that is expected to occur and a *planner* determines the steps which would occur in the audit trail.

More general applications in the field of distributed system management have been developed so far. Softbots(Etzioni *et al.* 1993) are planning-based AI agents using software tools and services through the World Wide Web. Softbots help the user to interact with real-world software environments such as operating systems or databases. In particular, *Rodney* is an Internet softbot representing a customizable intelligent assistant for Internet access. It generates and executes plans to achieve given goals, and learns from its experience. The planner used by Rodney adopts two approaches in order to deal with incomplete knowledge: it interleaves planning and execution of sensing actions when possible, otherwise it plans for all contingencies (Golden 1997). Occam(Kwock & Weld 1996) is a Softbot representing an information gathering engine for the Internet. It is based on a heuristic planner that builds suitable action sequences to obtain the information required.

Conclusion and Future Work

Automated security management is a challenging task in distributed systems. We have proposed a planning approach for an IDS reacting component. The planner is implemented in the Constraint Logic Programming language ECL^iPS^e (ECRC 1992). The proposed solution represents a flexible alternative to the static definition of intrusion countermeasures, it allows the management of new situations and unexpected events, it relies on the current system state by performing a sort of monitoring activity (through dynamic knowledge acquisition) while planning is in progress, is easily extensible by defining new declarative actions, and copes with the problem of intrusion false alarms. We are currently testing the system on real distributed environments in order to tune the threshold between complexity of basic planner actions and the real time response requirements of the applications.

As it is now, the planner does not allow to fully cope with dynamic knowledge: the retrieved knowledge is always referred to the initial state since the plan has not been yet executed. Precisely, once any information is retrieved the planner relies on it as a snapshot of a portion of world state assuming that it would not change until plan execution. However, we are currently implementing an ICS_based mechanism to verify action preconditions before any action execution and effects after their execution. If something goes wrong a repair activity is triggered in order to recovery from inconsistent states of the system, relying on backup actions when needed. Re-planning is then needed in order to achieve the original goal.

We are also investigating how the planner can tackle dynamic information at real time thus avoiding replanning activity.

Finally we are considering to exploit the planner also for dealing with concurrent tasks. Suppose that an attack triggers a plan processing and that during the computation of the plan, another attack is detected. It could be that some actions needed to recover from the former attack are also useful to deal with the latter. We are investigating the possibility of monitoring the system during the planning process and, when needed, add further goals to the *Agenda*, so that the planner can tackle both goals at the same time and likely produce an unique final plan.

We are aware that the definition of a completely automatic and autonomous IDS is still a long way off, but we think that planning represents a little step toward this task.

Acknowledgments

Authors' work is partially supported by Hewlett Packard Laboratories of Bristol-UK (Internet Business Management Department). We would like to thank M.Boari, E.Lamma and P.Mello for useful comments and discussion.

References

Barruffi, R., and Milano, M. 1998. Interactive constraint satisfaction techniques for information gathering in planning. *Proc. of ECAI'98.*

Barruffi, R.; Lamma, E.; Mello, P.; and Milano, M. 1998. Planning with incomplete and dynamic knowledge via ICS. *AIPS Workshop on Integrating Planning Sheduling and Execution in Dynamic and Uncertain Env. - AAAI Press.*

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. of AIPS94.*

ECRC. 1992. *ECL^iPS^e User Manual Release 3.3.*

Etzioni, O.; Levy, H.; Segal, R.; and Thekkath, C. 1993. OS agents: Using AI techniques in the operating system environment. Technical report, Univ. of Washington.

Garvey, T., and Lunt, T. 1991. Model based intrusion detection. In *Proc. of 14th National Computer Security Conference.*

Golden, K., and Weld, D. 1996. Representing sensing actions: The middle ground revisited. In *Proc. of KR'96.*

Golden, K. 1997. *Planning and Knowledge Representation for Softbots.* Ph.D. Dissertation, University of Washington.

ISS. 1998. *RealSecure User Manual Release 3.0.*

Jaffar, J., and Maher, M. 1994. Constraint logic programming: a survey. *Journal of Logic Programming - Special Issue on 10 years of Logic Programming.*

Joslin, D., and Pollack, M. 1995. Passive and active decision postponment. In *Proc. of EWSP'95.*

Kahn, C.; Porras, P.; Staniford-Chen, S.; and Tung, B. 1998. A common intrusion detection framework. *Proc. of Information Survivability Workshop.*

Kambhampati, S. 1996. Using disjunctive orderings instead of conflict resolution in partial order planning. Technical report, Dept. of Comp. Science and Eng. - Arizona State Univ.

Kumar, S., and Spafford, E. 1995. A software architecture to support misuse intrusion detection. In *Proc. of the 18th National Information Security Conference.*

Kwock, C., and Weld, D. 1996. Planning to gather information. Technical report, Dept. of Comp. Science and Eng. - Univ. of Washington.

Lever, J., and Richards, B. 1994. parcplan: a planning architecture with parallel actions, resources and constraints. In *Proc. of 8th ISMIS.*

Lunt, T. 1993a. Detecting intruders in computer systems. In *Proc. of the Conference on Auditing and Computer Technology.*

Lunt, T. 1993b. A survey of intrusion detection techniques. *Computers and Security.*

Myers, K. 1996. A procedural knowledge approach to task-level control. In *Proc. of AIPS'96.*

Olawsky, D., and Gini, M. 1990. Deferred planning and sensor use. In *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.*

R.Cucchiara; M.Gavanelli; E.Lamma; P.Mello; M.Milano; and M.Piccardi. 1999. Constraint satisfaction and value acquisition: why we should do it interactively. In *Proceedings of IJCAI'99.* To appear.

Smaha, S. 1998. Haystack: An intrusion detection system. In *Proc. of Fourth Aerospace Computer Security Applications Conference.*

Tate, A.; Drabble, B.; and Dalton, J. 1994. Reasoning with constraints within O-Plan2. Technical report, AI Applications Inst. - Univ. of Edinburgh.

Teng, H.; Chen, K.; and Lu, S. 1990. Security audit trail analysis using inductively generated predictive rules. In *Proc. of the 11th National Conference on Intelligence Applications.*

Tyson, M.; Moran, D.; Berry, P.; Blei, D.; Carpenter, J.; and Lang, R. 1998. DERBI: Diagnosis, Explanation and Recovery from Break-Ins. In *Adaptive Architecture Workshop.*

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15:27–61.

Yang, Q., and Chan, A. 1994. Delaying variable binding commitments in planning. In *The 2nd International Conference on AI Planning Systems (AIPS).*

Yang, Q. 1992. A theory of conflict resolution in planning. *AIJ* 58:361–392.