International Journal of Communication Networks and Security

Volume 1 | Issue 1 Article 11

April 2011

A Temporal Logic Based Approach to Multi-Agent Intrusion Detection and Prevention

Paritosh Das
Indian Institute of Techology Roorkee- 247667, India, pdas.rke@gmail.com

Rajdeep Niyogi
Indian Institute of Techology Roorkee- 247667, India, rajdpfec@iitr.ernet.in

A Temporal Logic Based Approach to Multi-Agent Intrusion Detection and Prevention

Paritosh Das, Rajdeep Niyogi
Department of Electronics and Computer Engineering
Indian Institute of Techology Roorkee
Roorkee- 247667, India
pdas.rke@gmail.com, rajdpfec@iitr.ernet.in

ABSTRACT
Collaborative systems research in the last decade have led to the development in several areas ranging from social computing, e-learning systems to management of complex computer networks.

Intrusion Detection Systems (IDS) available today have a number of problems that limit their configurability, scalability or efficiency. An important shortcoming is that the existing architectures is built around a single entity that does most of the data collection and analysis. This work introduces a new architecture for intrusion detection and prevention based on multiple autonomous agents working collectively. We adopt a temporal logic approach to signature-based intrusion detection. We specify intrusion patterns as formulas in a monitorable logic called EAGLE. We also incorporate logics of knowledge into the agents. We implement a prototype tool, called MIDTL and use this tool to detect a variety of security attacks in large log-files provided by DARPA.

Keywords
Multi-agent systems, Intrusion Detection, Intrusion Prevention, temporal logic, intelligent security, Alternating-time Temporal Epistemic Logic**.**

I INTRODUCTION

An Intrusion Detection System (IDS) is a computer program that attempts to perform ID by either signature-based or anomaly-based, or a combination of techniques. An IDS should ideally perform its task in real time. An Intrusion Prevention System (IPS) is a security component that has the ability to detect attacks and prevent the malicious behavior to succeed.

Few commercially available IDSs include Haystack [2], Midas [3] and Asax [4]. There are certain drawbacks in these system as given in [1]. Key drawbacks are i) scalability is limited, ii) central analyzer is a single point of failure, iii) it is difficult to add capabilities to the IDS, iv) the probabilities of false positives and false negative are too high and v) inability to detect distributed coordinated attack.

Thus the objective now is to make the intrusion detection systems more intelligent. Some proposed means of achieving such intelligence include use of neural networks, rule-based networks, genetic algorithms, human-like immunology systems [5] and Markov chains.

There are also approaches that used agents in achieving intrusion detection system. For example Balasubramaniyam et al [1], have used a hierarchical structure for intrusion detection. There is also immune-based multi-agent intrusion detection system by Wang et al [5]. Ontology based multi-agent intrusion detection systems by Isaza et al [6].

In this paper, we adopt a temporal logic based approach to signature-based intrusion detection. We use multi-agents in this framework. We also incorporate logics of knowledge into the agents.

We specify the intended behavior of intrusion attacks as temporal formulas, and monitor the system execution to check if a system event violates the formula. If the observed execution violates the formula then an intrusion has occurred. We also make use of autonomous agents to increase the scalability of our system and also to remove many other drawbacks of existing IDS. For our IPS we incorporated logics of knowledge into the agents. Thus our system can not only detect attacks but also can take remedial actions against those attacks.

Our approach to intrusion detection is motivated by how the use of multi-agent system has increased the performance of IDS when compared to standard centralized IDS [1]. Further motivations come from the success of runtime verification [7], the goal of which is to use light-weight formal methods for system monitoring.

We use EAGLE, introduced in [8], for specifying attacks pattern. We use a monitoring algorithm and process each event with the EAGLE formula and then modify the formula to store the result of the event. If, after any event the modified formula becomes false, an intrusion has occurred. Thus signature-based IDS is achieved by the careful use of temporal logic.

We show that our approach is effective by specifying several types of attacks and by monitoring those using MIDTL. We perform offline monitoring using the large log-files made available by DARPA [9]. The specified attacks have

been successfully detected. The experiments suggest that the proposed approach using multi-agents is better than existing IDSs.

We achieve Intrusion Prevention by incorporating logics of knowledge into the agents. The agents have a knowledge base within them in which they store the remedial actions to be taken against a certain attack. The agents can gain experience about wide variety of attacks by communicating with other agents present in different hosts.

II BACKGROUND AND MOTIVATION

A Use of Temporal Logic for Intrusion Detection

Signature-based approaches have low false alarm rate, but would fail to detect attacks that differ even slightly from the given signature. Signature-based approach has been popular because of their accuracy and simplicity. For signature-based system there have been several approaches to intrusion detection. For example, Roger et al. in [10] have used temporal logic and model-checking based approach to detect intrusion. In this paper we use monitoring logic called EAGLE [8] to achieve signature-based intrusion detection. EAGLE based approach seems to be simpler to describe. When using EAGLE we can express the attack pattern with real-time constraints. EAGLE supports recursively defined temporal formulas, which can be parameterized by data expressions. Once an attack has been specified by EAGLE formula *ö*, we can then monitor the system events against the formula. We use a monitoring algorithm and process each event with the EAGLE formula and then modify the formula to store the result of the event. Whenever the formula becomes false, we know that intrusion has occurred.

We use agents for intrusion detection. We distribute the task to detect intrusion into parts and distribute among the agents. Information about system events obtained from relevant log-files is captured by Sensor agents. A Transceiver agent then merges the events from various sources by timestamp. It can also filter the events, so only appropriate events can reach the analyzer agent. Analyzer agent then monitors these events against a given formula, and if the formula becomes false it informs Reaction agent. Reaction agent with the help of Communication agent sends alert messages to Monitor agent, which raises an intrusion alarm.

B Knowledge aspect of Agents

An interesting area of research in multi-agent systems is to understand the knowledge aspects of agents. A fact like 'agent A knows a fact *Õ*', is simple not enough, but also to express the fact that 'agent A knows that agent B knows *Õ*' and so on.

In our approach, we use a temporal logic that incorporates knowledge operators [11]. This logic is called Alternating Temporal Epistemic Logic (ATEL) [12], and is an extension of the Alternating Temporal Logic (ATL) of Alur et al [13]. ATL is a novel generalization of Computational Tree Logic (CTL) [14] in which the path quantifiers of CTL are replaced by *cooperation modalities*: the ATL formula $\langle\langle\Gamma\rangle\rangle$ à, where is a group of agents, expresses the fact that can cooperate to eventually bring about j. The CTL path quantifiers A ("on all paths. . . ") and E ("on some paths. . . ") can be expressed in ATL by the cooperation modalities ("the empty-set of agents can cooperate to. . . ") and ("the grand coalition of all agents can cooperate to. . . "). ATEL extends ATL by the addition of operators for representing knowledge. As well as operators for representing the knowledge of individual agents, ATEL includes modalities for representing what "everyone knows" and common knowledge.

In this section we design an architecture for our IDS and IPS system that uses agents for data collection and analysis. It also employs a hierarchical structure to allow for scalability. It also incorporates logics of knowledge into the agents.

A The Architecture

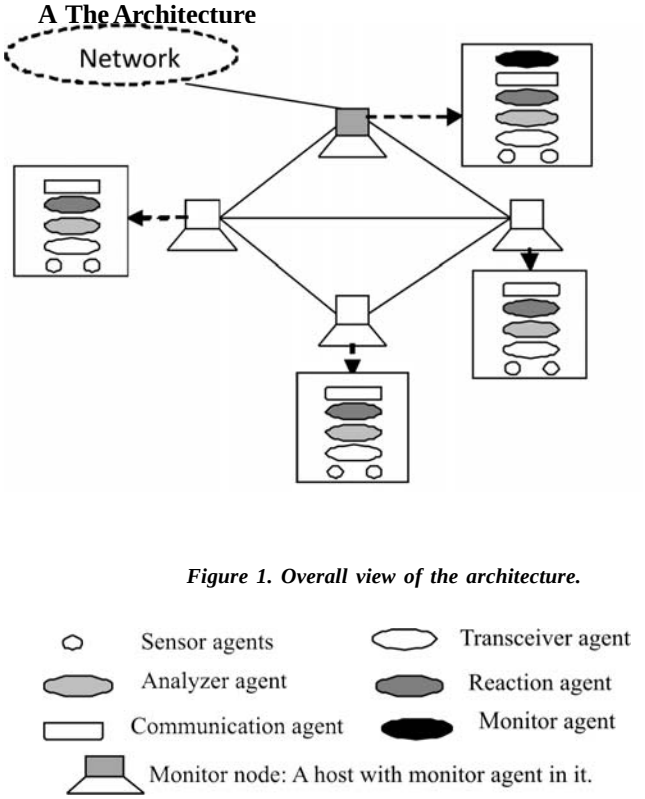

Figure 1. Overall view of the architecture.

The overall architecture of the network is shown in Figure. 1. This figure shows a subnet with four hosts. The subnet is connected to a larger network of similar type. The hosts in the subnet may or may not be fully connected. The figure also shows different types of agents present in a host. Each host in the network has MAS architecture. One particular host in the network has agents like Sensor agents, Transceiver agent, Analyzer agent, Reaction agent and Communication agent. Few hosts have Monitor agent, which does a higher level correlation and detect intrusions that involve several hosts. Such hosts are known as Monitor Node. Together with all these agents, they form a multi-agent community.

The sensor agents, transceiver agent, analyzer agent does the data collection and data analysis part. As we will soon see in section 4, analyzer agent uses temporal logic in detecting intrusions. Known attack patterns are expressed as formulas in a monitorable logic called EAGLE [8]. Analyzer agent uses a monitoring algorithm that matches specifications of the absence of an attack with system execution log, if the specification is violated; analyzer agent informs the findings to the reaction agent. Reaction agent has a knowledge base, so it does appropriate changes with its findings. It may update its knowledge base with new findings or it can do some useful correlation. With the help of communication agent, it informs a monitor agent about the intrusion. Monitor agent may be present in the same host or different host. Either way monitor agent does a higher level correlation with its own knowledge base and generates an alarm. Monitor agent helps in making the system scalable. It informs the network administrator about the intrusion. Network administrator can then take the necessary action. Thus this architecture presents an Intrusion Detection System. Reaction agent and Monitor agent has a knowledge base. It incorporates logics of knowledge in itself with the help of ATEL (Alternating Temporal Epistemic Logic) [12] as we will see in section 5. As a result of this Reaction agent can take remedial actions of its own without the need to inform Monitor agent or Network administrator. Thus this architecture can also perform as an Intrusion Prevention System.

In the following section we describe each component in greater detail.

B Overview

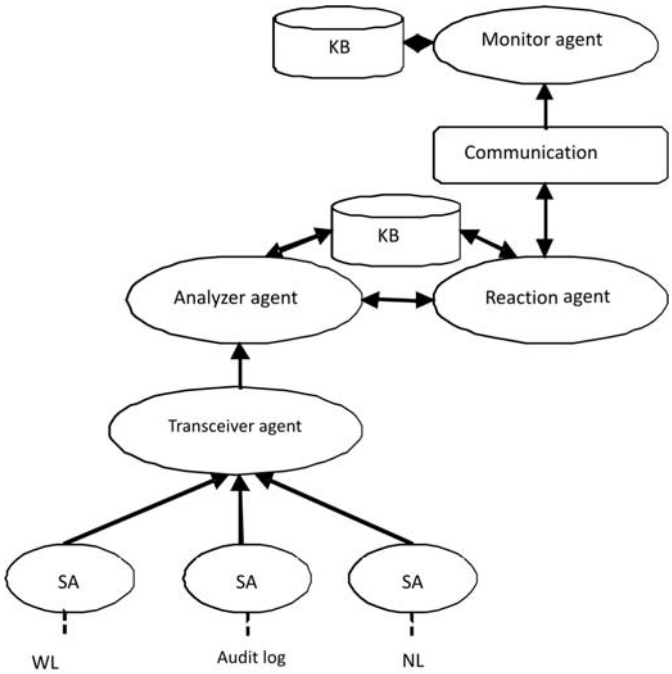

Figure 2. Multi-agent architecture

KB – Knowledge Base SA – Sensor Agent

WL – Webserver log NL – Network Log

Intelligent multi-agents for IDS and IPS architecture is shown in Figure. 2. Sensor agents, transceiver agent, analyzer agent, reaction agent and communication agent are present in all the hosts of the network. Few of the hosts in the network have monitor agents within them. This monitor agent has user interfaces by which they can directly contact the administrator of the network. All these agents are autonomous in nature. Some agents may or may not need data produced by other agents to perform their work.

The figure shows the multi-agent architecture inside a host. The output from sensor agents is fed into transceiver agent. The results of transceiver agent are given to analyzer agent. Analyzer agent and reaction agent work together. Reaction agent communicates with reaction agent or monitor agent of other host with the help of communication agent.

Reaction agent is mainly responsible for establishing IPS in this architecture. Prevention mechanisms are taken by a host with the help of reaction agent's knowledge base. Certain preventive measures like reconfiguring the firewall, removing a subnet from the network or by simply terminating a session are undertaken by the reaction agent. In section 5, we will learn about prevention mechanism in greater detail. With the help of this prevention mechanism, a reaction agent need not contact monitor agent whenever an intrusion takes place. Reaction agent can take care of the intrusion by itself. There is also no need to contact the administrator of the network.

The knowledge base of reaction agent is shared by the analyzer agent. Analyzer agent can read certain remedial actions against few attacks, for example the blocked IP address list. So, whenever an IP address from the blocked lists tries to create a session, analyzer agent informs reaction agent about it and that particular connection is immediately closed off

C An Example scenario

Figure. 3 shows a subnet with four hosts. The subnet is connected to a network. Host 4 has a monitor agent whereas the other host does not. Let us call Host 4 as monitor node.

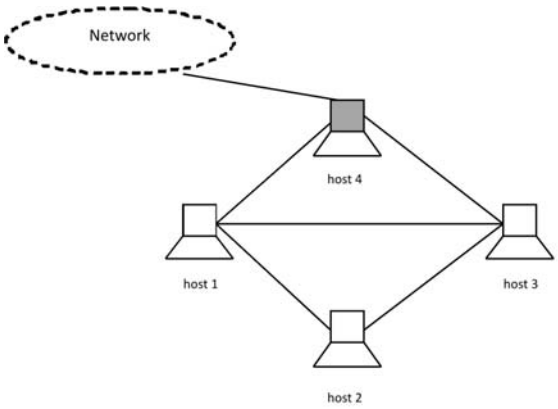

Figure 3. A subnet with four hosts.

Suppose analyzer agent of host 1 detects an attack. The reaction agent of host 1 first searches its own knowledge base for a remedial action against this attack. If there is an action available, reaction agent of host 1 performs the action, for example terminating a session or closing off a connection. Thus host 1 can take care of the attack all on its own.

If suppose there is no action available against the attack in its knowledge base, reaction agent of host 1 contact a monitor node of its subnet so that it can inform it of the attack, in this case it is host 4. Reaction agent of host 1 sends alert message about the attack by communicating with the monitor agent of host 4 either directly or indirectly. Monitor agent of host 4 upon receiving the alert message generates an alarm and informs the network administrator about the attack. The administrator takes appropriate actions.

Host 4 which has a monitor agent also does a higher level correlation of the entire subnet involving all four hosts. If it detects an attack it takes similar actions as above. If there are no remedial actions available in its knowledge base it generates an alarm and informs the network administrator.

IV DETECTING INTRUSIONS USING TEMPORAL LOGIC

In this section we see how analyzer agent detects intrusions. Analyzer agent as discussed above uses a temporal logic approach to signature-based intrusion detection. The idea is to specify the behavior of different intrusion attacks as temporal formulas, and monitor the system execution to see if it violates the formula. If the observed execution violates the formula then an intrusion has occurred.

A EAGLE: A monitoring logic

The intrusion attacks are specified formally in the logic EAGLE [8]. EAGLE contains a small set of operators. The three temporal operators used are next-time (O), previous-time (™"), and concatenation ( ). Rules can be parameterized with formulas and data-values.

Syntax

A specification *S* consists of a declaration part *D* and an observer part *O*. *D* consists of zero or more rule definitions *R*, and *O* consists of zero or more monitor definitions *M*, which specify what to be monitored. Rules and monitors are named (*N*). A rule definition R is preceded by a keyword indicating whether the interpretation is maximal or minimal.

$$
\begin{array}{ll}
S & ::= D\ O \qquad\qquad D ::= R^* \qquad\qquad O ::= M^* \\
R & ::= \{\underline{\max} \mid \underline{\min}\}\ N(T_1\ x_1, \ldots, T_n\ x_n) = F \\
M & ::= \underline{\mathrm{mon}}\ N = F \\
T & ::= \underline{\mathrm{Form}} \mid \textit{primitive type} \\
F & ::= \textit{expression} \mid \underline{\mathrm{true}} \mid \underline{\mathrm{false}} \mid \neg F \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid F_1 \rightarrow F_2 \mid \\
 & \quad \bigcirc F \mid \odot F \mid F_1 \cdot F_2 \mid N(F_1, \ldots, F_n) \mid x_i
\end{array}
$$

Semantics

Given a trace *ó* and a specification *D O*, satisfaction is defined as follows:

$$ó \mid\!= D\ O \text{ “ iff “ } (\mathrm{mon}\ N = F) \text{ “ } O \,.\, ó, 1 \mid\!=_D F$$

That is, a trace satisfies a specification if the trace, observed from position 1 (the first state), satisfies each monitored formula. The definition of the satisfaction relation $\mid\!=_D$ †" (*Trace* × **nat**) × Form, for a set of rule definitions *D*, is presented below, where $0\ d" i\ d" n + 1$ for some trace $ó = s_1 s_2 \ldots s_n$.

$$
\begin{array}{ll}
\sigma, i \models_D expression & \text{iff } 1 \le i \le |\sigma| \text{ and } evaluate(expression)(\sigma(i)) == true \\
\sigma, i \models_D \underline{\text{true}} & \text{iff } \sigma, i \not\models_D \underline{\text{false}} \\
\sigma, i \models_D \neg F & \text{iff } \sigma, i \not\models_D F \\
\sigma, i \models_D F_1 \wedge F_2 & \text{iff } \sigma, i \models_D F_1 \text{ and } \sigma, i \models_D F_2 \\
\sigma, i \models_D \bigcirc F & \text{iff } i \le |\sigma| \text{ and } \sigma, i+1 \models_D F \\
\sigma, i \models_D \odot F & \text{iff } 1 \le i \text{ and } \sigma, i-1 \models_D F \\
\sigma, i \models_D F_1 \cdot F_2 & \text{iff } \exists j \text{ s.t. } i \le j \le |\sigma|+1 \text{ and } \sigma^{[1,j-1]}, i \models_D F_1 \text{ and } \sigma^{[j,|\sigma|]}, 1 \models_D F_2 \\
\sigma, i \models_D N(\overline{F}, \overline{P}) & \text{iff } \begin{cases} \text{if } 1 \le i \le |\sigma| \text{ then:} \\ \quad \sigma, i \models_D B[\overline{f} \mapsto \overline{F}, \overline{p} \mapsto evaluate(\overline{P})(\sigma(i))] \\ \quad \text{where } (N(\underline{\text{Form}}\ \overline{f}, \overline{T}\ \overline{p}) = B) \in D \\ \text{otherwise, if } i = 0 \text{ or } i = |\sigma|+1 \text{ then:} \\ \quad \text{rule } N \text{ is defined as } \underline{\max} \text{ in } D \end{cases}
\end{array}
$$

We refer to [8] for detailed semantics.

Monitoring Algorithm

We describe here the computation mechanism used to check if an EAGLE formula is satisfied by a sequence of events. Our assumption is that the expressions of an EAGLE formula are specified with respect to the fields of the event record. At every event the algorithm evaluates the monitored formula on the event and generates another formula. At the end of the event sequence, the value of the evolved formula is determined; if the value is true the formula is satisfied by the event sequence, if it is false then the formula is violated.

The evaluation of a formula F at an event $s = ó(i)$ is transformed into another formula $F' = eval\ (F, s)$ with the property that $ó$, $i \models F$ if and only if $ó$, $i + 1 \models F'$, where eval is the monitor function. At the end of the trace we compute the Boolean function $value(F)$, where F is the evolved formula, such that $value(F)$ is true if and only if $ó$, $|ó| + 1 \models F$ and false otherwise. Thus for a given trace $ó = s_1 s_2 \ldots s_n$ and an Eagle formula F, $ó$ satisfies F if and only if $value(eval\ (\ldots eval\ (eval\ (F, s_1), s_2) \ldots, s_n)) = true$. The details of the algorithm can be found in [8].

V Different Types of Attacks and its Specification

In this section, we present few attack signatures using the logic EAGLE which can be used to monitor the system. The attacks classes and its descriptions are taken directly from [15]. The specifications of few attacks like Dictionary, Port-Sweep attack were given in [16].

Denial of Service Attacks

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service (or DoS) attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon.

SYN Flood

A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the "tcpd" server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out.

In order to detect this attack, we need to look at network events from a log created by tcpdump. The formula for absence of this attack is given by:

Check(int *f*, long *i1*, int *p*) = (*f* = *flag*) Ù (*i1* = *ip1*) Ù (*p* = *port*)
Count(int *f*, long *i1*, int *p*, int *a*, int *b*) = (*a* d" *b*)
® ((Check(*f*, *i1*, *p*) ® O Count(*f*, *i1*, *p*, *a* + 1, *b*))
Ù (¬Check(*f*, *i1*, *p*) ® Count(*f*, *i1*, *p*, *a* , *b*)))
NoSYNFloodAttack = Always(Count(*flag*, *ip1*, *port*, 1, 15))

In above example, flag would be "S" if the SYN bit is set. *ip1* is the destination ip address, port is a particular port number. The upper limit of the number of SYN packets that can be sent to a particular port is 15. *a* is the current number of SYN packets sent to a particular port, *b* is the upper limit. *f* holds the flag value and *i1* holds the destination ip address. Whenever a SYN packet is sent to the same port, the value of a is increased by one. If it goes beyond 15 then it is a SYN Flood attack.

Teardrop

The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments.

In order to detect this attack, we need to look at network events from a log created by tcpdump. The formula for absence of this attack is given by:

TearNext(int *i1*, int *s1*, int *o1*) = (*i1* = *i*) ® (*o1* > *s*) Ù (*s1* < *s*)
Tear(int *i*, int *s*, int *o*) = (*o* = 0) ® TearNext(*id1*, *size1*, *offset1*)
NoTeardropAttack = Always(Tear(*id*, *size*, *offset*))

We can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF == 0), with a positive offset greater than N and with a payload of size less than N. *id*, *size*, *offset* is the ip-fragment id, fragment size and its offset.

User to Root Attacks

User to Root exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit.

Eject

In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems.

A host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument.

In order to detect this attack, we need to have access to the host's audit logs. Auditing can be turned on by running Sun's Basic Security Monitoring (BSM) software.

Eject() = (*event_type* = "execve") Ù (*path* = "/usr/bin/eject")

Arg(int *n*) = (*n* d" 50)

NoEjectAttack = Always(Eject()® Arg(*num*))

In above formula, *num* means the number of arguments given when invoking the eject program. We can get this from BSM audit log. If the number of arguments exceeds 50 then we have an Eject attack. *event_type* is the current event the log is recording.

Ffbconfig

Ffbconfig is same as the Eject Attack. Due to insufficient bounds checking on arguments in the volume management library, it is possible to overwrite the internal stack space of the ffbconfig program.

A host-based intrusion detection system can look for the invocation of the command "/usr/sbin/ffbconfig/" with an oversized argument for the "-dev" parameter.

Ffbconfig() = (*event_type* = "execve") Ù (*path* = "/usr/bin/eject") Ù (*param* = "-dev")

Arg(int *n*) = (*n* d" 50)

NoFfbconfigAttack = Always(Ffbconfig()® Arg(*num*))

Remote to User Attacks

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network, but who does not have an account on that machine, exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies.

Dictionary

The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords. An attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords.

In order to detect this attack, we need to have access to the host's audit logs. We can get this from BSM audit log.

LoginError( ) = (type = *login*) Ù *failure*

LoginAgain(long i) = (*i* = ip) Ù LoginError( )

Check(long *t*, long *s*, int *c*, long *i*, int *d*) = (*time* – *t* < *s*)

® ((LoginAgain (*i*) ® (*c* d" *d* Ù O Check(*t*, *s*, *c* + 1, *i*, *d*)))

Ù (¬ LoginAgain (*i*) ® Check(*t*, *s*, *c*, *i*, *d*)))

NoDictionaryAttack = Always(LoginError( ) ® Check(*time*, 8000, 1, *ip*, 3))

In above rule, the arguments *t*, *s*, *c*, *i*, and *d* represent the rule invocation time, the timeout period, the current number of unsuccessful-guesses count, the source IP address doing the guess, and the threshold count. An attack occurs when the number-of-guess count *c* from the IP address *i* exceeds the threshold count *d* within the timeout period *s*. The monitor NoDictionaryAttack asserts that whenever there is a failure of login from an IP address then eventually within time 8000 the number of login-failures from the same IP address must be less than or equal to 3.

Port-Sweep Attack

A port sweep is a surveillance scan through many ports on a single network host. Each port scan is essentially a message sent by the attacker to the victim's port and elicits a response from the victim that indicates the port's status. The aim of the attacker is to determine which services are supported on the host, and use this information to exploit vulnerabilities in these services in the future.

PortScan(long *i*1, long *i*2, Set *S*) = (*i*1 = *ip*1) Ù (*i*2 = *ip*2) (*port* " *S*)

Check(long *t*, long *s*, int *c*, long *i*1, long *i*2, Set S, int *d*) = (*time* – *t* < *s*)® ((PortScan(*i*1, *i*2, *S*) ® (*c* d" *d* Ù Check(*t*, *s*, *c* + 1, *i*1, i2, *S* U {port}, *d*))) Ù (¬ PortScan(*i*1, *i*2, *S*)

® Check(*t*, *s*, *c* + 1, *i*1, i2, *S*, *d*)))

NoPortSweepAttack = Always(Check(*time*, 100, 1, *ip*1, *ip*2, { *port* }, 10)) The arguments *t*, *s*, *c* and *d* in the Check rule are the initial time, the timeout period, the frequency count, and the threshold count. The parameterized Check rule asserts that the number of port scans between a source and destination IP (*i*1 and *i*2) address pair never exceeds a certain threshold *d* within time *s*. Note that in the rule Check we add every new port number scanned to the set *S* of all port numbers that are scanned within time *s*.

VI EPISTEMIC PROPERTIES OF AGENTS

In this section we see how reaction agent takes preventive measures against intrusion attacks, how the logic of knowledge is incorporated in an agent. Reaction agent keeps the remedial action of certain attacks in its knowledge base. It also maintains certain knowledge about its neighboring agents. Let us consider a group of agents where some agents would like to share a common knowledge. This can be expressed using alternating temporal epistemic logic (ATEL) [12]. ATEL is alternating-time temporal logic (ATL) [13] extended to capture knowledge modality. We omit the details of the logics due to space constraints.

Figure 5. A subnet with three hosts.

Smurf Attack

The attack and its EAGLE specifications were discussed in section 4.2. Let us consider a network as shown in Figure. 5.

Suppose analyzer agent of host 4 has detected a smurf attack. It has been asked to host 4 to broadcast ICMP request message with host 1 as sender. Host 1 is the victim here. If certain actions are not taken immediately, network will get congested with all the ICMP echo replies being sent to host 1. Reaction agent of host 4 does the following remedial action: It tries to communicate with the sender of ICMP request message which is host 1 here in this case, either directly or indirectly. Thus host 4 communicates with its neighbors to communicate with host 1 and find out whether host 1 indeed sent an ICMP request message. In above case host 4, host 3 and all other neighbors of host 4, if it exists, collaborates to find out if host 1 sent an ICMP request message.

$\langle\langle\ 4,2,3\ \rangle\rangle$ *Did host 1 sent an ICMP echo request message?*

After reply from host 1 (the reply being sent by the reaction agent of host 1), host 4 now keeps a table in its knowledge base about host 1 that he did not sent any ICMP message. It can be represented as

$$K_4\,(reply = \text{no}) \rightarrow (\text{state}_{\text{ICMPmsg}}(\text{host1}) = \text{false})$$

Let this formula be represented as φ. Thus now there exist a group knowledge, where everyone in the group knows φ. $E_G\tilde{O}$, where G being host 2, host 3 and host 4 in this case. Now anytime in future if such similar attack occurs, reaction agent of host 4 looks into its knowledge base and finds out that the ICMP state of host 1 is still false and so it does not sends any ICMP echo reply thereby preventing the network from congestion.

If host 1 wants to send an ICMP request message, it first contacts its neighbors which in turn contacts other hosts of the network and informs them to change the ICMP state of host 1 in the table. Thus again there exist a group knowledge, where everyone in the group knows the fact that host 1 indeed want to sent a ICMP request message anytime in the future.

VII IMPLEMENTATIONS AND RESULTS

A MIDTL: Multi-Agent Intrusion Detection system based on Temporal Logic

A monitoring engine called EAGLE FLIER, is written in Java and is available as a library. The engine EAGLE FLIER converts the EAGLE formulas into Java classes and then it can be used for monitoring events. EAGLE FLIER can be downloaded from the authors of EAGLE [8].

In our tool MIDTL, we didn't use EAGLE FLIER instead we have converted the attack formulae into Java programs. Attacks like Dictionary and Eject are converted into Java programs. Data structures have been assigned to certain attacks. For example, Dictionary attack java program maintains a HashMap of String arrays for its function. The system MIDTL is written in Eclipse which is a multi-language software development environment.

All the agents in MIDTL have been implemented as user-level threads. Two agents never communicate using methods but only by exchanging messages. When there is no work to do, an agent goes to sleep, whenever there is work an agent wakes up the required agent.

B Results

We perform offline monitoring using MIDTL with the standard DARPA Intrusion Detection Evaluation data set [16]. The large log-files made available by DARPA are used exclusively for the task of evaluating intrusion detection systems. In our experiment we used the data sets provided in the 1998 offline Intrusion Detection Evaluation plan. There are two types of logs which are available for analysis. The tcpdump logs and Sun Basic Security Module (BSM) audit logs. We use the BSM log which contains audit information describing system

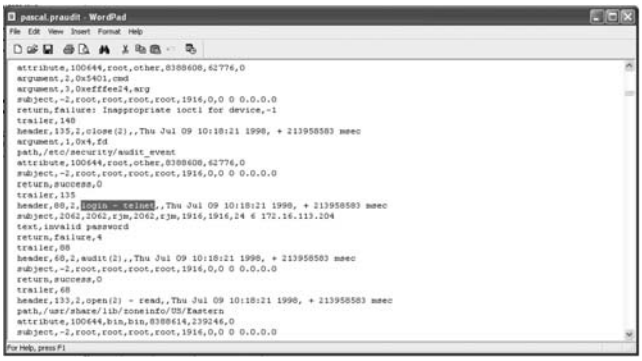

Figure 6. Snapshot of a BSM audit log.

calls made to the Solaris kernel. The size of typical BSM audit log varies from 200 – 300MB.

Figure. 6. shows a typical BSM audit log. The audit trail shows a number of events generated by the system. Every event starts with a *header*. In addition to other information *header* reports the event_type and also timestamp of that particular event. *Header* is followed by *subject* which explains the user's related information. Each record ends with *return* which reports the status of the event – success or failure. *Return* is usually followed by a *trailer.* Suppose for Dictionary attacks, we are interested only in "login – telnet" events.

MIDTL reads various log files through Sensor agent and sends relevant events to Analyzer agent for processing. Sensor agent sometimes filters the raw data, such as for Dictionary attack only *login* events are considered. Fig. 7. shows a filtered log. Analyzer agent then uses the above data structure and a monitor function to monitor the system events of the filtered log. For every relevant event monitor function is called iteratively and it modifies the data structure accordingly. Whenever any event causes to exceed a certain threshold, an alarm is generated. Analyzer agent then communicates this message to Monitor agent through Communication agent. Monitor agent then displays a warning message about the threat. The network administrator can then take the necessary action.

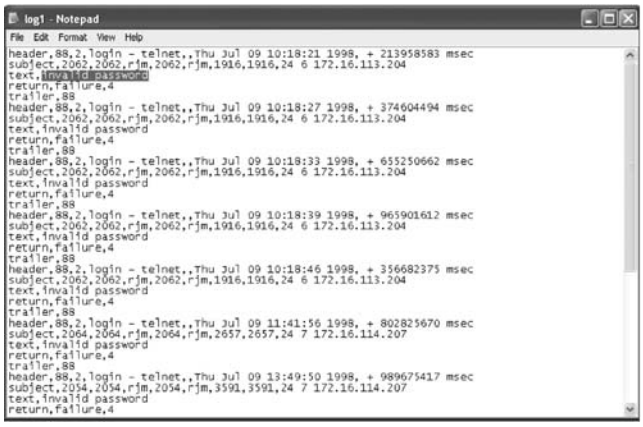

Figure 7. A filtered log.

We have implemented and tested our tool against the Dictionary attack and Eject attack. We ran our experiment on 1.70 GHz Pentium M laptop with 512MB RAM. Using MIDTL we were able to simulate the behavior of an intrusion detection system which passively processes events from our host offline. Our experiments successfully detected all the Dictionary and Eject attacks in the logs. Whenever an attack is detected, MIDTL displays a warning message to the user. The message contains the name of the attack, and the ip-address of the attacker. Figure. 8. shows the output of the MIDTL system.

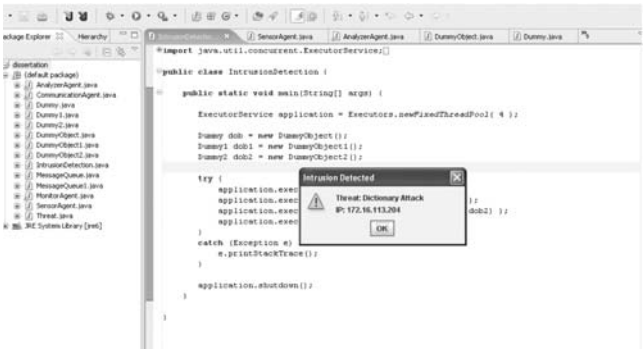

Figure 8. Output of the system.

The current version of MIDTL does only the work of an Intrusion Detection system. Prevention measures are to be incorporated in our future work. The functionality of Reaction agent was not implemented in MIDTL. Thus, it is the

responsibility of the user or network administrator to take necessary actions against such attacks.

VIII CONCLUSIONS AND FUTURE WORK

We have designed an architecture for an intelligent Intrusion Detection and Prevention System using multi-agents. We also employed a hierarchical structure to allow for scalability. We showed that our architecture can remove certain limitations of existing IDS. We used temporal logic approach to signature-based intrusion detection. Using temporal logic formulas to specify the intended behavior of security-attacks coupled with multi-agents, we obtained an intelligent Intrusion Detection System. Our tool MIDTL successfully detected well-known attacks from large event-logs made available by DARPA. We also incorporated logics of knowledge into the agents and using it we obtained an intelligent Intrusion Prevention System.

Future work can focus on making the agents more informative by knowledge acquisition, knowledge distribution and sharing. We can further reduce the false-positive and false-negative error rates by using hybrid techniques which combines intelligent models like genetics algorithms or fuzzy logic.

REFERENCES

[1] Balasubramaniyan, J. S. , Garcia-Fernandea, J. O., Isacoff, D., Spafford, E. and Zamboni, D.: An Architecture for Intrusion Detection using Autonomous Agents. http://citeseer.nj.nec.com/balasubramaniyan98architecture.htm1

[2] Smaha, S.E.: Haystack: an intrusion detection system. In: *Proceedings of Aerospace Computer Security Applications Conference* (1988) 37–44

[3] Habra, N., Charlier, B.L., Mounji, A., Mathieu, I.: Expert system in intrusion detection: A case study. In: *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD (1988) 74–81

[4] Habra, N., Charlier, B.L., Mounji, A., Mathieu, I.: Asax: Software architecture and rule-based language for universal audit trail analysis. In: *Deswarte, Y., Quisquater, J.-J., Eizenberg, G. (eds.) ESORICS, LNCS*, vol. 648. Springer, Heidelberg (1992) 435–450

[5] Dian Gang Wang, Tao Li, Sun Jun Liu, Gang Liang and Kui Zhao.: An Immune Multi-Agent System for Network Intrusion Detection. In*: ISICA, LNCS 5370*, (2008) 436-445

[6] Isaza, G. A., Castillo, A. G. and Duque, N. D.: An Intrusion Detection and Prevention Model Based on Intelligent Multi-Agent Systems, Signatures and Reaction Rules Ontologies. In*: 7th International Conference on PAAMS*. (2009) 237-245

[7] Sen, K., Vardhan, A., Agha, G. and Rosu, G.: Efficient decentralized monitoring of safety in distributed systems. In: *Proceedings of 26th International Conference on Software Engineering*, Edinburgh, UK, (2004) 418–427

[8] Barringer, H., Goldberg, A., Havelund, K. and Sen, K.: Rule-based runtime verification. In: *Proceedings of 5th International Conference on Verification, Model Checking and Abstract Interpretation*. Lecture Notes in Computer Science, Vol. 2937. Springer-Verlag, Venice, Italy, (2004) 44–57

[9] MIT Lincoln Laboratory. DARPA intrusion detection evaluation.

http://www.ll.mit.edu/IST/ideval/

[10] Roger, M. and Goubault-Larrecq, J.: Log auditing through model-checking. In: *14th IEEE Computer Security Foundations Workshop. IEEE* (2001)

[11] Fagin, R., Halpern, J., Moses, Y., Vardi, M.: *Reasoning about Knowledge.* MIT Press, Cambridge (1995)

[12] van der Hoek, W., Wooldridge, M.: Tractable multiagent planning for epistemic goals. In: *Proceedings of AAMAS* (2002)

[13] Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. In*: Proceedings of 38th IEEE Symposium on Foundations of Computer Science* (1997)

[14] Clarke, E.M., Grumberg, O. and Peled, D.A.: Model Checking. MIT Press, (1999)

[15] Kendall, K.: *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems*, Master's Thesis, Massachusetts Institute of Technology, (1998)

[16] Naldurg, P., Sen, K. and Thati, P.: A Temporal Logic Based Framework for Intrusion Detection. *FORTE. LNCS*. Vol. 3235. (2004) 359-376