Constructing Multi-Layered Boundary to Defend Against Intrusive Anomalies: An Autonomic Detection Coordinator

Zonghua Zhang, Hong Shen
School of Information Science
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Nomi, Ishikawa 923-1211, Japan
{zonghua, shen}@jaist.ac.jp

Abstract

An autonomic detection coordinator is developed in this paper, which constructs a multi-layered boundary to defend against host-based intrusive anomalies by correlating several observation-specific anomaly detectors. Two key observations facilitate the model formulation: First, different anomaly detectors have different detection coverage and blind spots; Second, diverse operating environments provide different kinds of information to reveal anomalies. After formulating the cooperation between basic detectors as a partially observable Markov decision process, a policy-gradient reinforcement learning algorithm is applied to search in an optimal cooperation manner, with the objective to achieve broader detection coverage and fewer false alerts. Furthermore, the coordinator's behavior can be adjusted easily by setting a reward signal to meet the diverse demands of changing system situations. A preliminary experiment is implemented, together with some comparative studies, to demonstrate the coordinator's performance in terms of admitted criteria.

1 Introduction

Most existing anomaly detectors (AD) intend to characterize a specific operating environment sufficiently well, with an expected false alert rate to be determined *a priori*, and most of them attempt to detect individual instantiations rather than classes of attacks, which limits their broader application. Usually, the first stage in establishing an anomaly detection model is to select the observable subjects (e.g., system call traces, network packet logs, command line strings), and construct the operating environments to characterize system normality. Due to their specific characteristics, different observations have different capabilities for characterizing system normality, and thus the constructed operating environment might limit their ability to discover some hidden intrusive attempts. For example, some attacks might be detected in system call stacks, whilst escaping from system call traces, and these phenomena also exist even for the same anomaly detection model. In this paper, we pay more attention to the effects of observations than to the specific detection techniques themselves. To achieve better performance in terms of broader detection coverage, higher detection accuracy, and fewer false alerts, we intend to develop a model to combine several observation-specific ADs with different properties. The basic assumption to support our work is that various operating environment could provide different kinds of normal and anomalous information for system characterization, and thus different ADs could create a consensus on the identification of anomalies, while intersecting their judgement on false alerts.

Another motivation for combining different ADs is to analyze and capture the "root-cause" of attack variants. It is well known that an attack might have different behaviors, and leave traces in various manners for the same system vulnerability. The combination of different ADs is expected to abstract specific or concrete behaviors sufficiently well to detect families of attacks rather than individual instantiations, thereby allowing for the detection of all the attack variants that attempt to exploit the same weakness. With those objectives in mind, we formulate several state-of-the-art ADs as a multi-agent Partially Markov Decision Process (POMDP). The proposed model, called an autonomic detection coordinator (ADC), is expected to work in a dynamic manner to find an optimal combination to adapt to the changing system situations with satisfactory performance in terms of predefined evaluation metrics. Moreover, the model could be easily extended to more complex situations, such as a network with distributed ADs, sensor networks, or wireless networks. Also, the probabilistic nature of the model guarantees it will work in a tolerant manner. Even

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

though one of the individual AD might fail to work properly, ADC can still collect enough intelligence from the other ADs, and make a correct decision based on their consensus. Therefore, adaptability, scalability, and dependability enrich the functionality of ADC significantly, compared to a single AD.

The paper is organized as follows: Section 2 reviews some related work. In Section 3, we formulate our model as a POMDP, and give a specific solution. In section 4, we implement the experiments and discuss the results; further issues are also discussed. A conclusion about our work is given in the last section.

2 Related Work

Han et al. [9] combined multiple host-based detection models using a decision tree to lower false alert rate with good performance on the detection accuracy. However, their detection models were established on the same layer (i.e., audit events and some related parameters and attributes), although the utilized information was different. Moreover, the decision tree essentially is a static approach, causing the model to be lack of adaptability, and the performance would deteriorate dramatically with the increasing number of elemental ADs. In addition, based on the observation that the human experts always attempt to design "root-cause" signatures that "combine" different attack characteristics in order to attain low false alarm rates and high attack detection rates, Giacinto et al. [8] proposed an approach to network intrusion detection by fusing multiple classifiers. In this work, intrusion detection essentially was formulated as a pattern recognition problem, and more efforts were paid to the comparative studies on the fusion approaches rather than the intrusion detection problem itself, specific analysis on the intrusion detection performance was not the emphasis either.

In addition, many cooperative intrusion detection models have been proposed to countermine distributed attacks by leveraging the information collected from distributed hosts, such as [15, 16], or to improve the accuracy of alarms by correlating different kinds of observations of multiple heterogeneous sensors [10, 14]. In those models, local agents or sensors are used to collect interesting events (from audit data, network packets, etc.) or alarm reports, and the distributed architectures provide various communication methods to exchange the locally detection information. Compared with the existing works, even though starting with similar motivation, our work focuses more on the anomaly detection model itself for correlating the anomaly reports from independent anomaly surrogates, and searches an optimal correlation on the system state from learning instead of relying on a predefined set of rules or events. The emphasis is on the the analysis of the model's anticipated behavior from a high level viewpoint, and the effects of the complementary correlation of different observations on revealing more anomalies.

In general, we envision a framework in which several levels of data analysis are used as the basis to be combined to yield a single but effective system normality characterization. We envision further an approach in which anomaly detection models are built on a fundamental understanding of their operating environments, and have the adaptability to respond to the diverse demands of various system situations. The hope is that a collection of simple surrogates based on specific observable subjects can cooperate and evolve into generic models with broader anomaly detection coverage and less false alerts.

3 Model Formulation

With the motivations presented in the section 1 and based on an observation-centric analysis on four typical ADs, we formulate our autonomic detection coordinator (ADC) as a POMDP model. A policy-gradient reinforcement learning algorithm is then used to search the optimal combination strategy based on the formulation.

3.1 Selection of Basic ADs

The selection of the individual ADs mainly takes into account following considerations:
1.) the trade-off between the computational cost and detection performance,
2.) since we use a host with UNIX OS as the experimental scenario, all the individual ADs are host-based, and work in different environments, or take advantage of different properties of the same observation,
3.) the population of the ADs should not be too large for the easy of control and analysis.

Minimum Cross Entropy (MCE) based on the occurrence frequencies of events is selected as one AD to operate with shell command lines; One-order Markov Chain is selected to be operated with Audit Events; Sequence, time-delay, embedding AD (STIDE) and K-Nearest-Neighbor (KNN) are selected as two elemental ADs to work with the system calls of privilege programs, but the properties they utilize are different. Table 1 shows the simple comparison between those four selected ADs, while the detailed description can be found in their respective references.

3.2 A General Formulation

Assume that each AD is an autonomous entity working in its own environment with uncertain perceptions, actions, and feedback, and each of them takes the action independently according to its local parameterized policy. our in-

Table 1. A Simple Comparison between Four Slected ADs

Anomaly Detectors	Observation	Main Property		Detection Cost
		Frequency	Ordering	
MCE [23]	Shell Command Lines	√		$O(N * m^2)$
Markov Chains [21,22]	Audit Events	√	√	$O(N * L)$
STIDE [7]	Local Ordering of System calls		√	$O(N * (L - w + 1))$
KNN [11]	Frequency of System calls	√		$O(N)$

'w' is the predefined window size, 'm' is the number of unique event

tegrated detection model ADC attempts to combine those independent entities in an optimal way, with the anticipated behavior to suppress false alerts and achieve broader detection coverage. It is worth noting that our main concern is the actions of independent ADs, rather than their inner detection mechanisms. The independent AD decides whether the ongoing activity is legal or malicious, and since each of them only works in its own environment, the true system state can only be indirectly observed through their respective detection measurement, and they must maintain the estimates of the true system state, therefore, the detection problem is partially observable for the entire system. Furthermore, the decision process is a Markov process, because the next state of the system is dependent only upon the current state and the previous decision. Thus, a partially observable Markov decision process is formulated here.

Formally, a POMDP contains several key parameters [1]:

- a finite state space of n distinct states, $S = \{1, 2, ..., n\}$ of the system
- a control space of m distinct actions, $U = \{1, 2, ..., m\}$ that are available to the detection policy
- an observation space of q distinct observations, $Z = \{1, 2, ..., q\}$
- a (possibly stochastic) reward $r(i) \in \mathbb{R}$ for each state

Specifically, the interactions between an independent AD and its operating environment includes a sequence of decision stages:

1. At time step i (discrete), the system in a particular state $s_i \in S$, and the underlying state emits an observation $z_i \in Z$ to the AD according to a probability distribution $\nu(s_i)$ over observation vectors.
2. The AD chooses an action $u_i \in U$ using a randomized policy, based on a probability distribution $\mu(z_i)$ over actions, with known z_i.
3. u_i determines a stochastic matrix $Pr(u_i) = [p_{ij}(u_i)]$, $p_{ij}(u_i)$ is the probability of making a transition from state s_i to state s_j under action u_i.
4. In every system state, the AD receives a reward signal r_i, while its aim is to choose a policy so as to maximize the long-term average of reward ($\mathbb{E}$ is the expectation operator),

$$\eta := \lim_{T \to \infty} \mathbb{E}[\frac{1}{T} \sum_{i=1}^{T} r_i]. \quad (1)$$

The above decision process shows that at each time step the AD sees only the observations z_i and the reward $r(i)$, while it has no knowledge of the underlying state space, how the actions affect the evolution of states, how the reward signals depend on the states, or even how the observations depend on the states. From another viewpoint, to each randomized policy $\mu(\cdot)$ and observation distribution $\nu(\cdot)$, the Markov chains for state transitions s_i and s_j are generated as follows:

$$s_i \in S \xrightarrow{\nu(s_i)} z_i \in Z \xrightarrow{\mu(z_i)} u_i \in U \xrightarrow{p_{ij}(u_i)} s_j \in S$$

In essence, all the above parameters can be organized into a family of action-dependent matrices: m $n \times n$ transition probability matrices F, m $n \times n$ observation probability matrices H, m $n \times n$ transition reward matrices G. $\nu(s_i)$ is essentially a $m \cdot n \cdot q$ known observation probability $Pr(z_i|s_i, u_{i-1})$, while $\mu(z_i)$ is a $q \cdot m \cdot m$ action probability $Pr(u_i|z_i, u_{i-1})$. In order to parameterize these chains, we parameterize the policies, so that $\mu(\cdot)$ becomes a function $\mu(\theta, z_i)$ of a set of parameters $\theta \in \mathbb{R}^k$ as well as the observation z_i. The Markov chain corresponding to θ has state transition matrix $P(\theta) = p_{ij}(\theta)$ given by $p_{ij}(\theta) = E_{z_i \sim \nu(s_i)} E_{u_i \sim \mu(\theta, z_i)} p_{ij}(u_i)$. Therefore, equation (8) can be achieved by the parameterized policy with θ:

$$\eta(\theta) := \lim_{T \to \infty} \mathbb{E}_\theta[\frac{1}{T} \sum_{i=1}^{T} r_i]. \quad (2)$$

As the detection process of each AD can be formulated as partially markov decision process, the ADC naturally can be modeled as a multi-agent POMDP. In the coordinator, several independent ADs with distinct operating environments are incorporated. Each of them sees a distinct observation vector, and has a distinct parameterized randomized policy that depends on its own set of parameters. If

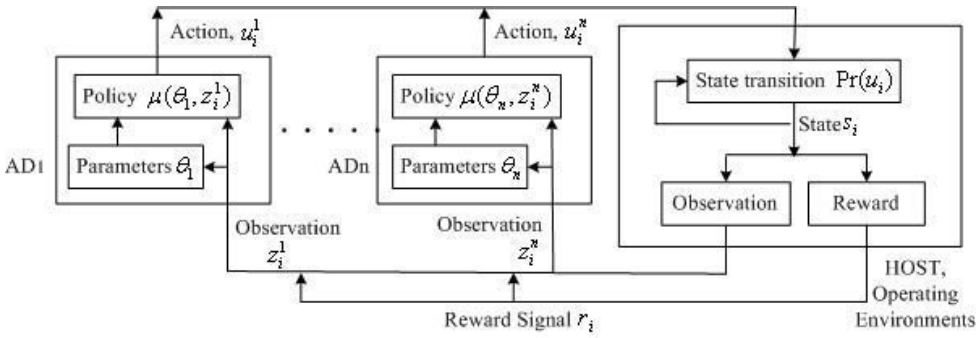

Figure 1. Architecture of the Autonomic Detection Coordinator

the collection of ADs is considered as a single AD, the individual observation vectors can be combined into a single observation vector, and similarly for the parameter vectors and action vectors, while the common goal of those ADs is to maximize the average reward. Effectively, each AD treats the other ADs as a part of the system, and updates its own policy while remaining oblivious to the existence of the other ADs. The only communication between these cooperating ADs is via the globally distributed reward signal, as shown in Figure 1. More formally, for the elemental ADs, the set of actions U contains the cross product of all the actions available to each AD, that is, $U = \{u_1 \times u_2 \times \cdots \times u_n\}$. Because the AD parameters are independent, each AD independently chooses actions that are combined to form the meta-action. For stochastic policies, the overall action distribution is the joint distribution of actions for each agent, $\mu(u_1, u_2, ...u_n|\theta_1, \theta_2, ...\theta_n, z_1, z_2, ...z_n)$.

3.3 A Specific Solution

In the formulated model, the policy of the ADC is affected by a concatenation parameter θ, while our aim is to find the parameter settings (the optimal control strategy) for all the ADs that maximizes the expected long-term average reward in equation (2). This is actually a kind of direct reinforcement learning problem, which is described in [4, 5].

Briefly, the algorithm learns to adjust the parameters θ of a randomized policy with observation z_i, and chooses actions according to $\mu(z_i, \theta)$. It involves the computation of a vector q_t at time step t, and it updates according to:

$$q_{t+1} = \rho \cdot q_t + \frac{\nabla \mu_{u_t}(z_t, \theta)}{\mu_{u_t}(z_t, \theta)} \tag{3}$$

where $\rho \in (0, 1)$, $\mu_{u_t}(z_t, \theta)$ is the probability of the action u_t under the current policy, and ∇ denotes the gradient with respect to the parameters θ. The vector q_t is an eligibility trace of the same dimensionality as θ; it is used to update the parameters, and guides the policy to climb the gradient of the average reward. Here, we intend to apply a multi-agent variant of the OLPOMDP algorithm [3], which has been applied to solve a routing problem by Tao et al [17], and a multi-neurons learning problem in the brain by Bartlett et al [2]. The OLPOMDP gives a simple way to compute an appropriate direction to update the parameters:

$$\theta_t = \theta_{t-1} + \triangle\theta = \theta_t + \tau_t \cdot r_t \cdot q_t \tag{4}$$

where the long-term average of the updates $\triangle\theta$ lie in the gradient direction $\nabla\eta(\theta)$, r_t is the sum of the rewards, and τ_t is the suitable size of the steps taken in parameter space. The key feature of the algorithm is that the only non-local information each detector needs is a global reward signal; detectors do not need to know any other information about the system state in order to climb the gradient of the global average reward.

Considering the specific characteristics of the host system, two assumptions need to be addressed to support the algorithm's application:

Assumption 1 *For every given θ, the system is ergodic (aperiodic, irreducible), and converges to a unique steady state $s_0 \in S$.*

Specifically, although the system's underlying states are unknown, it will return to a steady state ultimately; that is, the right-hand-side of equation (2) is independent of the system starting state, and converges with probability 1 over all possible reward sequences $\{r_i\}$.

Assumption 2 *For the POMDP-based ADC which is controlled by multiple independent ADs, the updates of equations (3) and (4) for the coordinator are equivalent to those that would be used by each AD.*

That is, if we let z_t^i denote the observation vector for AD $i, i = 4$, u_t^i denote the action it takes, and θ^i denote its parameter vector, the update equation (4) is equivalent to the individual update equations,

$$\theta_t^i = \theta_{t-1}^i + \tau_t \cdot r_t \cdot q_t^i \tag{5}$$

where $\tau_1, \tau_2, \cdots > 0$, $\sum_{t=0}^{\infty} \tau_t = \infty$, and $\sum_{t=0}^{\infty} \tau_t^2 < \infty$, while the vectors $q_t^i \in \mathbb{R}^k$ are updated according to

$$q_{t+1}^i = \rho \cdot q_t^i + \frac{\nabla \mu_{u_t^i}(z_t^i, \theta^i)}{\mu_{u_t^i}(z_t^i, \theta^i)} \tag{6}$$

where ∇ denotes the gradient with respect to the AD's parameters θ^i.

In addition, to cast the independent AD in the POMDP model, a formally definition is given as follows:

Definition 1 *All the ADs have no knowledge about the exact system states, in some sense, $|S|$ is infinite; the observation set Z=\{Normal, Malicious\}, and the action set U=\{Observe, Alert\} according to the specific detection algorithm.*

Suppose an activity (local or remote) happens at time step t, some or all of the four ADs will receive different observation streams independently in their own operating environments; assume $\sigma(\cdot)$ is a general form of the ADs' decision rule, which partitions the infinite measurement space into discretely different decision regions, with each region corresponding to one of a finite number κ (according to the definition 1, $\kappa = 2$) of possible output observations z_t. Given a measurement $\chi_0 \in \mathbb{R}$ on a measurement stream ℓ, AD i makes a decision with the decision rule parameterized by a threshold value λ_ℓ^i, as follows:

Definition 2 *For every measurement stream ℓ, there is a decision rule $\sigma_\ell : \mathbb{R} \to \{0, 1\}$ of the form*

$$\sigma_\ell(\chi_0) = \begin{cases} 0, & \chi_0 \le \lambda_\ell^i \\ 1, & \chi_0 > \lambda_\ell^i \end{cases}$$

where output "0" denotes the 'Normal' observation and "1" the 'Malicious' observation. Corresponding actions 'Observe' and 'Alert' are taken according to the observation.

From the definition, for the ADs, there is a direct mapping from *observations* to *actions*:

$$\{Normal, Malicious\} \to \{Observe, Alert\}$$

therefore, the process from observation to action essentially is deterministic. The parameter θ of ADC is a concatenation of parameters $\lambda^i (i = 1, 2, 3, 4)$, and it is a row vector with form $\theta = (\lambda^1, \lambda^2, \lambda^3, \lambda^4)$. Furthermore, it is worth noting that λ^i is only the threshold that determines the distance between normal activities and anomaly activities, while the action of ADs are also affected by other inner parameters. For instance, for STIDE, the window size of system call sequences w, the locality frame count (LFC) L can also be adjusted to impact the observation. For MCE, the length of command blocks L is also an adjustable parameters (but actually according to the login session). While for the Markov Chain detector, the length of sequences L is regarded as the parametric variables which affects the similarity between two sequences. However, because most of those inner parameters are related to the training phase, we do not include them into the concatenation parameter vector θ here.

The next consideration is to to derive the second term of the right-hand side in equation (6) for every independent AD. Since it is difficult to parameterize the underlying detection schemes with λ^i, in order to make the ADs trainable and save computational cost, we assume a general probabilistic model for the behavior of AD. Specifically, if we assume p is the *a prior* detection probability of AD, the probability of detecting n anomalies among N activities is:

$$P_p(n|N) = \binom{N}{n} p^n (1-p)^{N-n} \tag{7}$$

taking the distribution as the function of the expected number of successful detections, $v = pN$, the equation becomes:

$$P_{v/N}(n|N) = \binom{N}{n} (\frac{v}{N})^n (1 - \frac{v}{N})^{N-n}$$

When $N \to \infty$, we have,

$$\begin{aligned} P_v(n) &= \lim_{N\to\infty} P_{v/N}(n|N) \\ &= \lim_{N\to\infty} \frac{N(N-1)\cdots(N-n+1)}{n!} \frac{v^n}{N^n} (1 - \frac{v}{N})^{N-n} \\ &= \frac{v^n e^{-v}}{n!} \end{aligned}$$

Obviously, $P_v(n)$ is a Poisson distribution. Hence, for the independent AD, its action u_t generally obeys the following rule (based on the fact that the number of anomalies is much smaller than that of normal activities, we describe the model of $Pr(u_t = 0)$ rather than $Pr(u_t = 1)$).

$$Pr(\text{Observe without Alarms}) = Pr(u_t = 0) = \varphi(\varepsilon_t) \tag{8}$$

where $\varphi(x) = e^{-x}$, and $\varphi(x) \in [0, 1)$, while $\varepsilon_t \in (0, \infty)$ is defined as:

$$\varepsilon_t = \frac{\lambda_t^i}{d_t^i(\ell)} \tag{9}$$

where λ_t^i is the threshold of ith AD at time instant t, while $d_t^i(\ell)$ denotes the measurement distance between ongoing observations ℓ and the normal patterns. Assumption 2 shows how to update the threshold λ_t^i in the direction that maximally increases the long-term average of the reward. From equation (9), we easily derive

$$\frac{\frac{\partial}{\partial \lambda_t^i} \mu_{u_t}}{\mu_{u_t}} = \begin{cases} \frac{\varphi'(\varepsilon_t)}{d_t^i(\ell)\varphi(\varepsilon_t)} = \frac{-1}{d_t^i(\ell)} & \text{if } u_t = 0 \\ \frac{-\varphi'(\varepsilon_t)}{d_t^i(\ell)(1-\varphi(\varepsilon_t))} = \frac{\varphi(\varepsilon_t)}{d_t^i(\ell)(1-\varphi(\varepsilon_t))} & \text{otherwise} \end{cases} \tag{10}$$

To complete the picture we need to define a performance measure for the detection result, which can be taken as a reward signal to guide the improvement of the general detection performance. As we know, in the anomaly detection domain, some or all of following cases might happen:

- $\mathcal{N}_n$, legal behavior is detected as normal
- $\mathcal{N}_a$, legal behavior is detected as anomaly
- $\mathcal{A}_n$, illegal behavior is detected as normal
- $\mathcal{A}_a$, illegal behavior is detected as anomaly

Based on those four cases, a natural performance metric can be defined as:

Definition 3 *Assume that during a particular time period Δt, m activities occurred, among those activities, $i \in \mathcal{N}_n$, $j \in \mathcal{N}_a$, $k \in \mathcal{A}_n$, and $l \in \mathcal{A}_a$, if we assign w_1, w_2, w_3, and w_4 to denote their respective weights, and $\alpha = w_1 \cdot i/m$, $\beta = w_2 \cdot j/m$, $\zeta = w_3 \cdot k/m$, $\delta = w_4 \cdot l/m$, a reward signal can be defined as $r_t = \alpha \cdot \delta - \beta \cdot \zeta$, while w_1, w_2, w_3 and w_4 is defined according to various system situation and security demands.*

Due to the nature of anomaly detection, and the fact that the number of normal activities is much larger than that of anomalies, we usually set $w_1 < w_3 < w_4 < w_2$.

In essence, the anticipated behavior of our autonomic detection coordinator is based on the consensus of meta-AD, and thus we have another assumption,

Assumption 3 *Given an ongoing activity happens in the host at time step t, logically, the POMDP-based ADC gets the report as follows:*

$$R_d^c = \bigcup_{i=1}^{n} R_d^i(\mathcal{A}_a), \qquad R_f^c = \bigcap_{i=1}^{n} R_f^i(\mathcal{N}_a)$$

where R_d^i is the report of AD i about the detected anomalies, while R_f^i is the report about the false alerts.

Based on our specific assumptions and definitions, a modified version of algorithm OLPOMDP [4] can be used to describe the independent AD i as follows:

Algorithm *Model of ADC meta-action*

Given:
Coefficient $\rho \in [0, 1)$,
Step size τ_0,
Initial system state s_0,
Initial thresholds of independent ADs λ_0^i, i.e., θ_t^i in concatenation vector θ.
begin
  for discrete time instant $t = 1, 2, \cdots$ **do**
    Get ongoing observations and their corresponding measurement stream ℓ.
    Generate action u_t^i according to the specific detection scheme and definition 2.
    The coordinator broadcasts the reward signal r_t.
    Update q_{t+1}^i according to equation (6) and (10):
      if the previous actions is "*Observe*" (i.e., $u_t = 0$)
        $q_{t+1}^i = \rho \cdot q_t^i - \frac{1}{d_t^i(\ell)}$.
      else
        $q_{t+1}^i = \rho \cdot q_t^i + \frac{\varphi(\varepsilon_t)}{d_t^i(\ell)(1-\varphi(\varepsilon_t))}$.
      end if
    Update θ_{t+1}^i according to equation (5):
      $\theta_{t+1}^i = \theta_t^i + \tau_t \cdot r_t \cdot q_{t+1}^i$.
  end for
end

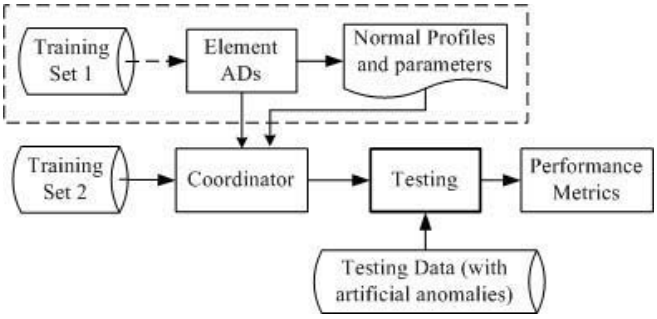

Figure 2. Experiment Procedure

Note that r_t in equation (6) is the sum of rewards that have been received, and q_t^i is a trace of the same dimensionality as θ_t^i, with $q_0^i = 0$; $\rho \in [0, 1)$ is a free parameter to control both the bias and the variance of the estimates produced by the algorithm. It has been shown that [2] provided the bias is sufficiently small, it will converge to a region of near-zero gradient, which thus can be extended to the multi-detector environment, and the algorithm does not need access to the underlying state and does not make use of recurrent states.

4 Performance Verification

This section describes the evaluation of our proposed ADC prototype, and the general evaluation procedure is shown in figure 2. Specifically, the procedure mainly includes following steps:

Step 1. To train the individual ADs with training data set 1, which only contains normal data, to get their initial parameters and create normal profiles in their respective operating environments.

Step 2.To train the ADC with training data set 2 (only pure normal data, or mixed with some known anomalies). This step can be combined with step 1 if some satisfied data with controllable property are available.

Step 3. After the ADC achieved a stable state through step 2, testing set (collected data with some artificial anomalies) is used to evaluate its performance it terms of detection accuracy and false alarms.

Table 2. Statistics of the Data Source

Data Category\Source		No. of Command Lines	No. of Audit Events	No. of Processes
Training Set (Normal)		5,600	62,100	640
Testing Set	Normal Data	5,640	70,780	690
	Masquerader	2127	850	272
	Other Attacks	*no trail*	*uncounted*	35

4.1 Experiment Scenario and Data Collection

An intrusion instance is exemplified in the following to show the operating scenario of our ADC. A keyboard masquerader or remote interloper takes control of a terminal/host, and then takes advantage of the legitimate user's privileges and access to system programs and data. The intruder may attempt to read or write access to private data, acquire unauthorized system privileges (or even abuse of legitimate privileges), and install some softwares such as *Trojan* for further malicious behavior. For the sophisticated intruder with knowledge of AD installed in target terminal, he might take some seeming legal tricks to surpass the detection coverage. In such activity, the intruder leaves trace data, in various forms, to victim terminal, such as shell command lines (especially for keyboard masquerader) with corresponding audit events, privilege processes with system calls, etc. The ADC is thus expected to detect those anomalies during the malicious attacks based on the trace data.

To the best of our knowledge, there is no true trace data in the open literature that meets our experimental demands. Therefore, we have to collect, combine and formulate our own experimental data with some particular considerations. For the sake of simplicity, all the basic ADs we employe are initialized with the parameters in their original literature; in other words, the first step is omitted in our experiment. To formulate training data set 2 and the testing data, we have collected normal activities ourselves for four weeks using the Solaris 8.0 operating system (SunOS release 5.0), mixed with several known typical host-based attacks.

We usually use text editor (*vi*, *ed*, etc.), compiler(*gcc, cc, etc.*), and some system programs(*ps, lpr, sendmail etc.*) on our machine SunBlade 1500. Excluding wrong commands and some noisy data, while keeping repeated ones, we obtained a total of 132,886 records of BSM audit data and 11,240 shell command lines (using the shell *.history* file to log all truncated commands without additional information), and these data were roughly averaged as part of pure training set and as testing set. Note that during the collection of shell command lines, we also recorded the corresponding audit events and executed processes in terms of system calls, as BSM provides the monitor of the execution of system calls by all processes launched by the user. However, considering the processes in user mode usually cannot harm the system security, we only recorded those processes in kernel model that require services from system kernel. In addition, it is well known that *buffer overflow*, *S/W security error*, *configuration error* and *DoS attacks* are several prevalent host-based attacks, so we injected several cases of them (audit data that contain labelled attacks), including 8 cases of *local buffer overflow* and 5 cases of *DoS*, into the testing data. Meanwhile, a small batch of another user's commands history (2127 audit events, 850 command lines, and 272 processes) were also added into the testing set as a masquerader trace data. Table 2 and table 3 shows the experimental data we used in detail.

Table 3. Attacks List in the Experiments

Attack Category	Attack Description	# of Cases
Masquerader	access to programs and data as an imposter by controlling the keyboard	850 commands
Buffer Overflow	*xlock* heap buffer overflow vulnerability	2
	eject buffer overflow vulnerability	3
	lpset buffer overflow vulnerability	3
DoS	Exhausting Disk Space (with *dd*)	2
	Exhausting the Memory	1
	Consumption of process table	2

4.2 Experimental Results and Analysis

All the basic ADs' initial parameters we used were directly derived from their original version (as shown in Table 4) without training. Thus, the parameter vector θ is a matrix with size $N \times M$, where M is the number of elemental ADs, N is the number of controllable parameters, and the Table 4 can be denoted as following in terms of θ_0, which is the initial state of the coordinator. But in actual experiment, we only adjust the first row of θ, i.e., $N = 1, M = 4$.

$$\theta_0 = \begin{pmatrix} 0.45 & 0.80 & 0.60 & 0.72 \\ 30 & 10 & 6 & 0 \end{pmatrix}$$

Table 4. Parameters of Basic ADs

* 'L' denotes Sequence Length, 'λ' is the threshold

	MCE	Markov Chain	STIDE	KNN
L	30	10	6(*LFC*=20)	variable
λ	0.45	0.80	0.6	0.72

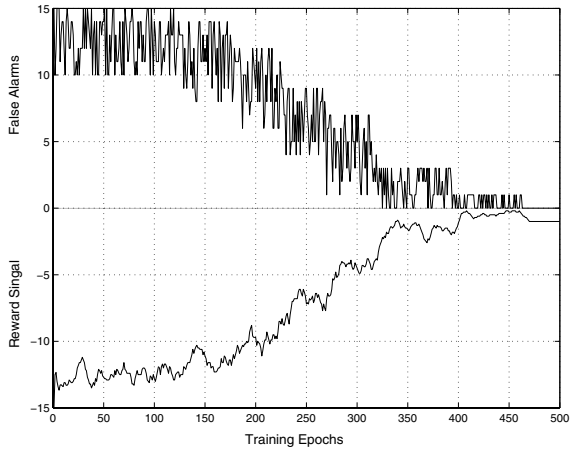

Figure 3. Reward Signal During Training

4.2.1 Training Procedure

The goal of the training procedure is to achieve an optimal control strategy of the ADC. As shown in table 2, all 5,600 command tokens were used to create a distribution-based behavioral model for MCE. Corresponding audit events and processes were also used to create normal profiles for Markov Chains, STIDE, and KNN respectively. Since the amount of the available data are limited, we used joint sets to train ADC, in detail, half of training data were interleaved with half of testing data (altogether 5,620 command tokens, 66,400 audit events, and 660 processes) to train the ADC. As every login session (i.e., from login to logout) contains about 30 command tokens, for simplicity, we used a constant window to partition command tokens, with corresponding audit events and system calls. Hence, a total $\lfloor \frac{5620}{30} \rfloor = 187$ commands blocks were available. Corresponding audit events and system calls that executed by processes were also extracted as input into respective ADs. The baseline of the ADC detection measurement is command blocks, which has no so exact mapping with their underlying audit events and processes. Therefore, Markov Chain, STIDE and KNN would generate a report sequence rather than a single report at every decision step, based on their respective detection measurement and parameters.

In this experiment, ADC makes decision at every command trace, and according to definition 3, since $\mathcal{A}_n$ and $\mathcal{A}_a$ would never appear in the normal training set, for a pursued probabilistic policy, the long-term expected average reward would be calculated based on the probabilities of two outcome occurring: $\mathcal{N}_n$ and $\mathcal{N}_a$. Hence r_t is simplified as $\alpha - \beta$, in addition, as the ADC gets reward signal at every decision step, r_t can be further simplified as $w_1 - w_2$. Specifically, at each time step, for the observation trace, if ADC takes action "*Observe*", reward signal is assigned 0, if the action is "*Alarm*", reward signal is assigned -1. A total reward signal is then calculated after one pass through the sequence data concatenated by observation traces (the ideal value should be 0). To simplify the consensus strategy, any false alarm reported by any elemental AD would led to the "*Alarm*" action of ADC, with penalty to all ADs.

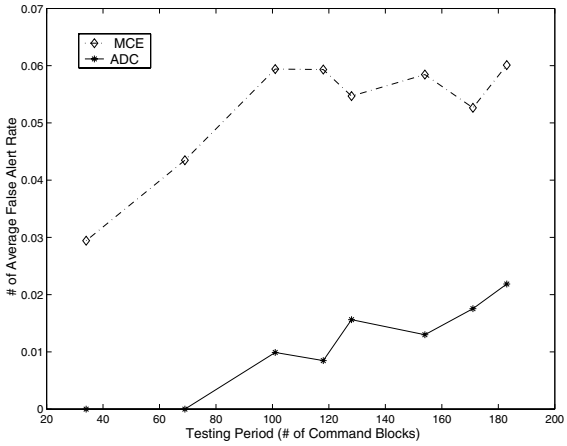

Figure 4. False Positive Rate on Testing Data

Figure 3 depicts the ADC's behavior during the training phase (with 500 training epochs, parameters ρ=0.90 and $\tau_1 = \tau_2 \cdots = 10^{-3}$). The upper part of the figure shows the changing of the number of false alarms in the training phase, and the lower part of the figure shows the average reward signal (to manifest the trend, ADC only considered the past 10 passes, i.e., T=10 in equation (2)). The figure shows clearly that the ADC had incrementally improved performance during the training phase, as the reward signal improves, on average, over time to an optimum. We found that after the 462th pass, there was no false alarm triggered. After being trained, the parameter vector of ADC θ is:

$$\theta = \begin{pmatrix} 0.42 & 0.84 & 0.69 & 0.79 \\ 30 & 10 & 6 & 0 \end{pmatrix}$$

4.2.2 Testing of False Alarms

To evaluate the capability of the ADC to suppress false alarms, we tested the trained ADC using the normal testing set in table 2. The testing data was also divided into 188 commands traces (each trace contains 30 command tokens), together with their underlying audit events and processes.

Table 5. Comparison of masquerade detection results between MCE and ADC

Methods	MCE	ADC
# of normal command traces	188	188
# of anomalous Command traces	28	28
Traces size	30	30
Hits(%)	71.43	82.14
Misses(%)	28.57	17.86
F.P.(%)	11.17	9.57
Total Detected	21	23

Figure 4 shows the relationship between the average false alarm rate (the number of false alerts over the number of command traces) and the number of command traces used for testing data. Since the ADC gives the report with the pace of each command trace, we compared its performance with that of MCE (with initial parameter), which also reports once on every command trace.

The figure shows that the ADC triggered less false alerts compared with MCE. ADC generated its first false alarm at the 101th command trace (i.e., F.P.=0.99%, first 94 command traces has been used to train ADC, thereby no false alarms were triggered until the 101th command trace). At the 183th command trace, MCE has generated 11 alerts, i.e., F.P.=6.01%, while the ADC only generated 4 alerts, F.P.=2.19%. We found that at the 128th and 171th command traces, MCE did not report false alarm, while ADC reported, which means that one of the other 3 ADs has made wrong actions. Although the analysis of other three ADs are helpful to insight into the story, we did not carry it out here, because of the intractable data partition and the lack of a compelling need to do. In addition, the parameter used by the MCE was directly derived from the ADC rather than by individual training, therefore, we can not rule out the possibility that the MCE might achieve better performance after being trained and parameterized carefully with another training dataset.

4.2.3 Detection of Common Exploits

First, we evaluated the ADC's masquerade detection performance. 850 command tokens (with underlying 2127 audit events and 272 processes) of another user were truncated into 28 command traces (each login session also contains 30 command tokens or so), and injected at randomly selected positions, without replacement, into the stream of original 188 command traces (a more complicated case is to inject the command traces into the command tokens instead of command traces; in such a case, the boundaries between the traces might generate uncontrollable false alarms). Meanwhile, the underlying audit events and processes that have been executed by the '*masquerader*' were also injected into the respective normal observation traces. The result is shown in Table 5, among total 216 command traces (188 normal + 28 anomalous), MCE detected 20 out of 28 anomalous command traces with a F.P. 11.17% by regulating the threshold to 0.38. After this detection spot, the F.P. raised sharply to 100% with a total 21 anomalous command traces being detected. While the trained ADC detected 23 anomalous command traces with a F.P. 9.57%.

Table 6. Detection Performance Comparison
('B' denotes Buffer overflow instance, 'D' denotes DoS instance)

	Hits(%)	F.P.(%)	Detected Attacks	Threshold
Markov Chain	84.62	4.35	8B+3D	0.88
STIDE	92.30	3.48	8B+4D	0.75
KNN	76.92	5.36	8B+2D	0.95
ADC	100.00	1.01	8B+5D	–

Second, the trained ADC was used to detect the injected attacks that shown in Table 3, and its performance was compared with that of the individual ADs. In our work, *detection accuracy* is defined as the ratio of the detected attacks to all the injected attacks (hidden in 35 intrusive processes). *false alert rate* is the ratio of the misreports to all the normal processes (total 690). To simplify the experiment while keeping the validity, we assumed that the false alerts would not be generated by those normal traces that have been used in the last experiment for testing false alerts, and the consensus strategy hence was adjusted as: any '*Alarm*' report from any individual ADs would cause the ADC to take '*Alarm*' action. The initial parameters used by the individual ADs were directly derived from the ADC, while to investigate the relationship between detection accuracy and F.P., we had to adjust them individually. Table 6 shows the detection result of the ADC, and the best trad-off between the detection accuracy and F.P. of the elemental ADs by adjusting respective thresholds (a higher detection performance would cause a dramatic increase of false alerts). Specifically, we have following observations:

- since the intrusive processes were injected into the normal processes without corresponding command traces, MCE always took action '*Observe*';
- the ADC detected all the injected attacks by combing the reports from elemental ADs, while its false alert rate was very low (i.e., 7 among 690 processes were misreported);
- all the ADs detected all the *buffer overflow* attacks, while some *DoS* attacks were not discovered.

5 Conclusion and Future Work

Based on the assumption that optimal combination of several observation-specific ADs may broaden detection coverage, suppress the false positive rate, and probably capture "root-cause" attacks, a POMDP model was formulated and a policy-gradient reinforcement learning algorithm was applied to tackle the delayed reward, partially observable, multi-agent learning problem. In next stage, we intend to collect more real trace data (and some artificial anomalies) to enrich the experiments. Some additional problems, such as computational cost, real-time response ability, and consensus efficiency, also need careful consideration. Furthermore, we will extend our work to the computer networks, to verify whether our ADC can detect distributed attacks with ADs locating in several dominated hosts. Anomalies in wireless networks or sensor networks are also expected to be detected through the optimal cooperation of location-centric ADs.

6 Acknowledgement

This research is conducted as a program for the "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for Promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology.

References

[1] Douglas Aberdeen, "A Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes", *National ICT Australia Report,* Canberra, Australia, 2003.

[2] Peter L. Barlett and Jonathan Baxter, "Hebbian synaptic modifications in spiking neurons that learn", *Technical report, Computer Sciences Laboratory,* RSISE, ANU, 1999.

[3] Jonathan Baxter and Peter L. Barlett, "Stochastic Optimization of Controlled Partially Observable Markov Decision Processes", *Proceedings of the 39th IEEE Conference on Decision and Control(CDC00)*.

[4] Jonathan Baxter and Peter L. Barlett, "Direct Gradient-Based Reinforcement Learning: I.Gradiment Estimation Algorithms", *Technical report*, ANU,1999.

[5] J. Baxter, L. Weaver, and P.L. Bartlett, "Direct Gradient-Based Reinforcement Learning: II.Gradiment Descent Algorithms and Experiments", *Technical report, Research School of Information Sciences and Engineering,* Australian National University, September 1999.

[6] Sung-Bae Cho and Hyuk-Jang Park. "Efficient anomaly detection by modeling privilege flows using hidden Markov model", *Computer and Security,* Vol No.1, pp 45-55,2003.

[7] S. Forrest, S.A. Hofmeyr,,& T.A. Longstaff, "A sense of self for UNIX processes", *In proceedings of 1996 IEEE Symposium on Security and Privacy*, Los Alamitos, CA: IEEE Computer Society Press.

[8] Giorgio Giacinto, Fabio Roli, Luca Didaci, "Fusion of multiple classifiers for intrusion detection in computer networks", *Pattern Recognition Letters* 24(2003) 1795-1803.

[9] Sang-Jun Han, and Sung-Bae Cho, "Combining Multiple Host-Based Detectors Using Decision Tree," *Artificial Intelligence,* LNAI 2903, pp.208-220, 2003.

[10] Joshua H., Dorene K.R., Larra T., Stephen T., "Validation of Sensor Alert Coorelators," *IEEE Security and Privacy,* pp46-56, 2003.

[11] Yihua Liao, V. Rao Vemuri, "Use of K-Nearest Neighbor classifier for intrusion detection", *Computers and Security*, Vol 21, No 5, pp439-448, 2002.

[12] Roy A. Maxion, "Masquerade Detection Using Enriched Command Lines", *International Conference on Dependable Systems & Networks:* San Francisco, CA, 22-25 June 2003.

[13] Roy A. Maxion, "Masquerade Detection Using Truncated Command Lines", *International Conference on Dependable Systems & Networks:* Washington, DC, 23-26 June 2002.

[14] Ning, P., Cui, Y., Reeves, D.S., and Ding X., "Techniques and Tools for Analyzing Intrusion Alerts," *ACM Transactions on Information and Systems Security,* Vol.7, No.2, May 2004, Pages 274-318.

[15] Porras, P.A., Neumann, P.G., "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proceedings of the 20th Natonal Information Systems Security Conference,*1997.

[16] Snapp, S.R., Smaha, S.E., Teal, D.M., Grance, T., "The DIDS (Distributed Intrusion Detection System) prototype", *the summer USENIX Conference,* San Antonio, Texas, USENIX Association (1992)227-233.

[17] Nigel Tao, Jonathan Baxter, Lex Weaver, "A Multi-Agent, Policy-Gradient approach to Network Routing", *18th International Conference on Machine Learning*, ICML 2001.

[18] Kymie M.C. Tan and Roy A. Maxion, ""Why 6" Defining the Operational Limites of stide, an Anomaly-Based Intrusion Detector," *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

[19] Kymie M.C. Tan, Kevin S. Killourhy, and Roy A. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits," *RAID 2002, LNCS 2516,* pp. 54-73, 2002.

[20] C. Warrender, S. Forrest and B. Pearlmutter. "Detecting Intrusion Detection Using System Calls: Alternative Data Models", *In Proceedings of 1999 IEEE Symposium on Security and Privacy*, pp133-145, Oakland, 1999.

[21] Nong Ye, Xiangyang Li, Qiang Chen, Syed Masum Emran, and Mingming Xu. "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data", *IEEE Transaction on Systems, Man, and Cybernetics-Part A:Systems and Humans,* Vol.31, No.4, July 2001.

[22] Nong Ye, Timothy Ehiabor and Yebin Zhang, "First-Order Versus High-Order Stochastic Models For Computer Intrusion Detection", *Quality and Reliability Engineering international*, 2002;18: 243-250.

[23] Dit-Yan Yeung, Yuxin Ding, "Host-based intrusion detection using dynamic and static behavioral models", *Pattern Recognition* 36 (2003) 229-243.