Novelty-Aware Attack Recognition – Intrusion Detection with Organic Computing Techniques

Dominik Fisch, Ferdinand Kastl, and Bernhard Sick

Computationally Intelligent Systems Lab, University of Passau, Germany
{fisch,kastl,sick}@fim.uni-passau.de

Abstract. A typical task of intrusion detection systems is to detect known kinds of attacks by analyzing network traffic. In this article, we will take a step forward and enable such a system to recognize very new kinds of attacks by means of novelty-awareness mechanisms. That is, an intrusion detection system will be able to recognize deficits in its own knowledge and to react accordingly. It will present a learned rule premise to the system administrator which will then be labeled, i.e., extended by an appropriate conclusion. In this article, we present new techniques for novelty-aware attack recognition based on probabilistic rule modeling techniques and demonstrate how these techniques can successfully be applied to intrusion benchmark data. The proposed novelty-awareness techniques may also be used in other application fields by intelligent technical systems (e.g., organic computing systems) to resolve problems with knowledge deficits in a self-organizing way.

1 Introduction

Organic Computing (OC) has emerged recently as a challenging research area dealing with future computationally intelligent systems that will be based on so-called self-x properties such as self-organization, self-optimization, self-configuration, self-healing, self-protection, or self-learning [1,2]. An example for such a system is an intelligent distributed system, e.g., a team of robots, a smart sensor network, or a multi-agent system. Often, the nodes of such a system have to perform the same or similar tasks, or they even have to cooperate to solve a given problem. Typically, these nodes know *how to observe* their local environment and this knowledge is represented by certain rules. However, many environments are dynamic. That is, new rules are necessary or existing rules become obsolete. Therefore, really intelligent nodes should adapt on-line to their environment by means of certain machine learning techniques.

In this article we focus on components—intrusion detection agents (IDA)—of a distributed intrusion detection system (DIDS). These IDA are able to analyze network traffic and to distinguish between "normal" network data (connections) and data originating from certain kinds of attacks (or tools used to prepare an attack). The rules that are used for that purpose are learned from sample data. The challenge is now that an IDA must be able to detect new kinds of attacks as well as new kinds of "normal" data which it had not seen before. This

M. Hinchey et al. (Eds.): DIPES/BICC 2010, IFIP AICT 329, pp. 242–253, 2010.

property is termed *novelty-awareness*. Then, the IDA must support a system administrator in creating new rules. That is, it must present an autonomously learned rule premise to the system administrator who must "label" that rule (i.e., determine an appropriate conclusion). Then, the new rule must be added to the existing rule system. In a future version of our DIDS, new rules will be exchanged between different IDA of the DIDS. Then, an IDA will be enabled to recognize attacks that it had not seen before.

The architecture of an IDA, the DIDS, a framework for large-scale simulations, and techniques for alert aggregation are described in [3,4,5] in more detail. Here, we focus on the novelty-aware attack recognition at the detection layer. This kind of novelty-awareness is very new to the field of intrusion detection. Thus, this article should be seen as a kind of proof of concept, where many of the components will be further improved in the future.

In the following, Section 2 briefly discusses related work in the field of DIDS, Section 3 lays the theoretical and methodological foundations of novelty-awareness, and Section 4 provides some experimental results. Finally, Section 5 summarizes the major findings and gives an outlook to our future work.

2 Related Work

Here, we briefly discuss related work in the field of DIDS. Information about terminology in the field of intrusion detection can be found in [6].

In most cases, the collaborative aspect of DIDS is found in the correlation of distributed data. Classical DIDS research is mostly focused on systems where agents located on network nodes aggregate data and send alerts to a central agent for correlation [7]. [8] proposes a p2p overlay network which allows correlation of attacks across domain borders with the goal of reducing the false positive rate and the reaction time. [9] proposes a system where different agents use different methods for attack detection (misuse or anomaly detection). [10] uses clustering at centralized nodes to perform correlation tasks. [11] describes a distributed fuzzy classifier, where distributed agents perform fuzzification of local data sources. A central fuzzy evaluation engine aggregates the agents findings and generates alerts according to pre-trained rules.

There are only a few proposals for systems which collaborate in some form to improve the performance of attack detection in the distributed agents. These systems are more closely related to our work. [12] describes an artificial immune system based DIDS where a primary IDS generates detectors (negative selection) and secondary IDS on the hosts perform detections and performance evaluation (clonal selection). [13] uses a genetic algorithm (island model) to train decision trees. Individual hosts form islands and work independently, but can exchange individuals among the different islands' gene-pools.

3 Theoretical and Methodological Foundations

In this section our proposed approach for realizing self-adaptive IDA is presented. First, we describe how classification knowledge is represented within the

agents. Then, we show how this kind of knowledge can be learned given a set of training data. Finally, techniques for adapting learned knowledge to changes in the environment are introduced.

3.1 Representation of Classification Knowledge

To classify network connections we use a probabilistic approach. That is, for a D-dimensional input sample $\mathbf{x}$ containing information about a specific connection (e.g., duration of the connection or number of transmitted packets) we want to compute the posterior distribution $p(c|\mathbf{x})$, i.e., the probabilities for class membership given an input $\mathbf{x}$. To minimize the risk of classification errors we then select the class with the highest posterior probability (cf. the principle of *winner-takes-all*). According to our previous publication [14], $p(c|\mathbf{x})$ can be decomposed as follows:

$$p(c|\mathbf{x}) = \sum_{j=1}^{J} \underbrace{\frac{\int_{\mathbf{x}\in\mathcal{R}_c} p(j|\mathbf{x})\mathrm{d}\mathbf{x} \cdot p(c)}{p(j)}}_{p(c|j)} \cdot \underbrace{\frac{p(\mathbf{x}|j)p(j)}{\sum_{j'=1}^{J} p(\mathbf{x}|j')p(j')}}_{p(j|\mathbf{x})} . \tag{1}$$

In this classification approach based on a so-called *mixture density model* $p(\mathbf{x})$, the conditional densities $p(\mathbf{x}|j)$ ($j \in \{1, \ldots, J\}$) are the components of the model, $p(j)$ is a multinomial distribution with parameters π_j (the mixing coefficients or rule "weights"), the $p(c|j)$ are multinomial conditional distributions with parameters $\xi_{j,c}$, and $\mathcal{R}_c$ is the (not necessarily connected) region of the input space associated with class c. That is, we have a classifier (rule set) consisting of J *rules*, where each rule j is described by a distribution $p(j|\mathbf{x})$ (which we call the *rule premise*) and a distribution $p(c|j)$ (which we call the *rule conclusion*). We can state that the former can be trained in an unsupervised way while class labels for patterns are needed for the latter. For a particular sample $\mathbf{x}'$, the values $p(j|\mathbf{x}')$ are called responsibilities (i.e., of the component for the sample).

Which kind of density functions can we use for the components? Basically, our D-dimensional input samples $\mathbf{x}$ describing network connections may have D_{cont} continuous (i.e., real-valued) dimensions and $D_{\text{cat}} = D - D_{\text{cont}}$ categorical ones. Without loss of generality we arrange these dimensions such that

$$\mathbf{x} = (\underbrace{x_1, \ldots, x_{D_{\text{cont}}}}_{\text{continuous}}, \underbrace{\boldsymbol{x}_{D_{\text{cont}}+1}, \ldots, \boldsymbol{x}_D}_{\text{categorical}}).$$

Note that we italicize x when we refer to single dimensions. The continuous part of this vector $\mathbf{x}^{\text{cont}} = (x_1, \ldots, x_{D_{\text{cont}}})$ with $x_d \in \mathbb{R}$ for all $d \in \{1, \ldots, D_{\text{cont}}\}$ is modeled with a multivariate *normal* distribution with center $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, i.e.,

$$\mathcal{N}(\mathbf{x}^{\text{cont}}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D_{\text{cont}}}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-0.5 \left(\Delta_{\boldsymbol{\Sigma}}(\mathbf{x}^{\text{cont}}, \boldsymbol{\mu})\right)^2\right) \tag{2}$$

with the distance measure (matrix norm) $\Delta_{\boldsymbol{M}}(\boldsymbol{v}_1, \boldsymbol{v}_2)$ given by $\Delta_{\boldsymbol{M}}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \sqrt{(\boldsymbol{v}_1 - \boldsymbol{v}_2)^T \boldsymbol{M}^{-1}(\boldsymbol{v}_1 - \boldsymbol{v}_2)}$. $\Delta_{\boldsymbol{M}}$ defines the *Mahalanobis distance* of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^{D_{\text{cont}}}$ based on a $D_{\text{cont}} \times D_{\text{cont}}$ covariance matrix $\boldsymbol{M}$.

For categorical dimensions we must extend our approach presented in [14] by means of multinomial distributions. We use a 1-of-K_d coding scheme where K_d is the number of possible categories of attribute $\boldsymbol{x}_d$ ($d \in \{D_{\text{cont}+1}, \ldots, D\}$). The value of such an attribute is represented by a vector $\boldsymbol{x}_d = (x_{d_1}, \ldots, x_{d_{K_d}})$ with $x_{d_k} = 1$ if $\boldsymbol{x}_d$ belongs to category k and $x_{d_k} = 0$ otherwise. Categorical dimensions are modeled by means of *multinomial* distributions. That is, for an attribute $\boldsymbol{x}_d \in \{\boldsymbol{x}_{D_{\text{cont}+1}}, \ldots, \boldsymbol{x}_D\}$ we use

$$\mathcal{M}(\boldsymbol{x}_d|\boldsymbol{\delta}_d) = \prod_{k=1}^{K_d} \delta_k^{x_{d_k}} \tag{3}$$

with a parameter vector $\boldsymbol{\delta}_d = (\delta_{d_1}, \ldots, \delta_{K_d})$ and the restrictions $\delta_{d_k} \geq 0$ and $\sum_{k=1}^{K_d} \delta_{d_k} = 1$.

We assume that the categorical dimensions are mutually independent and that there are no dependencies between the categorical and the continuous dimensions. Thus, the component densities $p(\mathbf{x}|j)$ are defined by

$$p(\mathbf{x}|j) = \mathcal{N}(\mathbf{x}^{\text{cont}}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \prod_{d=D_{\text{cont}}+1}^{D} \mathcal{M}(\boldsymbol{x}_d|\boldsymbol{\delta}_{j_d}). \tag{4}$$

3.2 Knowledge Acquisition Using Sample Data

How can the various parameters of the classifier be determined? For a given training set $\boldsymbol{X}$ with N input samples $\mathbf{x}_n$ and corresponding target classes it is assumed that the $\mathbf{x}_n$ are independent and identically distributed. First, the parameters of $p(\mathbf{x})$ are computed in an unsupervised manner. Let $\boldsymbol{\theta}$ be the overall set of model parameters consisting of all $\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j, \boldsymbol{\delta}_{j_d}$, and π_j. Then, the *likelihood function* of the parameters $\boldsymbol{\theta}$ given the data $\boldsymbol{X}$ is defined by

$$p(\boldsymbol{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{x}_n|\boldsymbol{\theta}). \tag{5}$$

We are searching the parameter setting that maximizes this function. In the case of a mixture density model it is not possible to evolve a closed formula for the optimization. However, by introducing the concept of *latent* (i.e., unobserved) variables, iterative methods can be used. For each sample $\mathbf{x}_n$ one of the J components is "responsible". To describe the "assignment" of samples to components, an additional latent random variable $\mathbf{z}_n$ is introduced for each sample. $\boldsymbol{Z}$ denotes the set of all latent variables.

In this work we perform model parameter estimation by means of a technique called variational Bayesian inference (VI) which realizes the Bayesian idea of

regarding the model parameters $\boldsymbol{\theta}$ as random variables whose distributions must be trained. This approach has two important advantages over other methods. First, the estimation process is more robust, i.e., it avoids "collapsing" components, so-called singularities whose variance in one or more dimensions vanishes. Second, VI optimizes the number of components by its own. For a more detailed discussion on Bayesian inference, and, particularly, VI see [15]. For the model described in the previous section we need the joint distribution of all random variables (i.e., observations, latent variables and model parameters) which can be decomposed into

$$p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}, \boldsymbol{\delta}) = p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\mu}, \boldsymbol{\Lambda}, \boldsymbol{\delta}) p(\boldsymbol{Z}|\boldsymbol{\pi}) p(\boldsymbol{\pi}) p(\boldsymbol{\mu}|\boldsymbol{\Lambda}) p(\boldsymbol{\Lambda}) \prod_{d=D_{\text{cont}+1}}^{D} p(\boldsymbol{\delta}_d). \quad (6)$$

where $\boldsymbol{\pi} = \{\pi_j\}, \boldsymbol{\mu} = \{\boldsymbol{\mu}_j\}, \boldsymbol{\Lambda} = \{\boldsymbol{\Lambda}_j\}$, and $\boldsymbol{\delta}_d = \{\boldsymbol{\delta}_{j_d}\}$. Note that for convenience we are using precision matrices $\boldsymbol{\Lambda}_j$ which are the inverses of the covariance matrices (i.e., $\boldsymbol{\Lambda}_j = \boldsymbol{\Sigma}_j^{-1}$). Unfortunately, this approach comes with an infeasible computational effort, and, thus, an approximation must be used. Therefore, it is assumed that the joint distribution of latent variables and model parameters p can be approximated by a function q that can be factorized as follows:

$$q(\boldsymbol{Z}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}, \boldsymbol{\delta}) = q(\boldsymbol{Z}) q(\boldsymbol{\pi}) \prod_{j=1}^{J} q(\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j) \prod_{d=D_{\text{cont}+1}}^{D} q(\boldsymbol{\delta}_{j_d}). \quad (7)$$

The distributions of the model parameters on the right hand side are called prior distributions and for an efficient computation their functional form must be chosen in a special way (so called *conjugate* prior distributions, cf. [15]). For the parameters $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$, a Gauss-Wishart distribution must be used as prior distribution [15], i.e.,

$$q(\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j) = \mathcal{N}(\boldsymbol{\mu}_j | \boldsymbol{m}_j, (\beta_j \boldsymbol{\Lambda}_j)^{-1}) \mathcal{W}(\boldsymbol{\Lambda}_j | \boldsymbol{W}_j, \nu_j) \quad (8)$$

where $\boldsymbol{m}_j, \beta_j, \boldsymbol{W}_j$, and ν_j are the parameters of the distribution that are determined during training (see below). The parameters $\boldsymbol{\pi}$ and $\boldsymbol{\delta}_{j_d}$ are assumed to be multinomially distributed and, thus, Dirichlet priors must be used, i.e.,

$$q(\boldsymbol{\pi}) = \text{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \quad \text{and} \quad q(\boldsymbol{\delta}_{j_d}) = \text{Dir}(\boldsymbol{\delta}_{j_d}|\boldsymbol{\epsilon}_{j_d}). \quad (9)$$

The corresponding parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\epsilon}_{j_d}$ are also infered during training. The VI is conducted iteratively by alternating between two steps. In the first step, the responsibilities $\gamma_{n,j}$ of components j for patterns $\mathbf{x}_n$ are evaluated:

$$\gamma_{n,j} = \frac{\rho_{n,j}}{\sum_{j'=1}^{J} \rho_{n,j'}}, \quad (10)$$

where

$$\ln \rho_{n,j} = \mathbb{E}[\ln \pi_j] + \frac{1}{2}\mathbb{E}[\ln |\boldsymbol{\Lambda}_j|] - \frac{D_{\text{cont}}}{2} \ln(2\pi)$$
$$-\frac{1}{2}\mathbb{E}_{\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j}[(\mathbf{x}_n^{\text{cont}} - \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j (\mathbf{x}_n^{\text{cont}} - \boldsymbol{\mu}_j)] + \sum_{d=D_{\text{cont}+1}}^{D} \sum_{k=1}^{K_d} x_{nd_k} \mathbb{E}[\ln \delta_{jd_k}]$$

with

$$\mathbb{E}[\ln \pi_j] = \psi(\alpha_j) - \psi\left(\sum_{j=1}^{J} \alpha_j\right), \quad \mathbb{E}[\ln \delta_{jd_k}] = \psi(\epsilon_{jd_k}) - \psi\left(\sum_{k=1}^{K_d} \epsilon_{jd_k}\right),$$

where $\psi(\cdot)$ is the Digamma function,

$$\mathbb{E}[\ln |\boldsymbol{\Lambda}_j|] = \sum_{i=1}^{D_{\text{cont}}} \psi\left(\frac{\nu_j + 1 - i}{2}\right) + D_{\text{cont}} \ln 2 + \ln |\boldsymbol{W}_j|,$$

and

$$\mathbb{E}_{\boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j}[(\mathbf{x}_n^{\text{cont}} - \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j (\mathbf{x}_n^{\text{cont}} - \boldsymbol{\mu}_j)] = \frac{D_{\text{cont}}}{\beta_j} + \nu_j (\mathbf{x}_n^{\text{cont}} - \boldsymbol{m}_j)^T \boldsymbol{W}_j (\mathbf{x}_n^{\text{cont}} - \boldsymbol{m}_j).$$

In the second step, the parameters of the prior distributions $q(\cdot)$ are adapted. With $N_j = \sum_{n=1}^{N} \gamma_{n,j}$ being the "effective" number of samples generated by component j, $N_{jd_k} = \sum_{n=1}^{N} \gamma_{n,j} x_{d_k}$ the "effective" number of samples belonging to category k in dimension d generated by component j and the statistics

$$\overline{\mathbf{x}}_j = \frac{1}{N_j} \sum_{n=1}^{N} \gamma_{n,j} \mathbf{x}_n^{\text{cont}}, \qquad \boldsymbol{S}_j = \frac{1}{N_j} \sum_{n=1}^{N} \gamma_{n,j} (\mathbf{x}_n^{\text{cont}} - \overline{\mathbf{x}}_j)(\mathbf{x}_n^{\text{cont}} - \overline{\mathbf{x}}_j)^T$$

the update formulas are given by

$$\alpha_j = \alpha_0 + N_j, \qquad \beta_j = \beta_0 + N_j, \qquad \nu_j = \nu_0 + N_j,$$

$$\epsilon_{jd_k} = \epsilon_0 + N_{jd_k}, \qquad \boldsymbol{m}_j = \frac{1}{\beta_j}(\beta_0 \boldsymbol{m}_0 + N_j \overline{\mathbf{x}}_j)$$

and

$$\boldsymbol{W}_j^{-1} = \boldsymbol{W}_0^{-1} + N_j \boldsymbol{S}_j + \frac{\beta_0 N_j}{\beta_0 + N_j} (\overline{\mathbf{x}}_j - \boldsymbol{m}_0)(\overline{\mathbf{x}}_j - \boldsymbol{m}_0)^T.$$

In the formulas we see some parameters indexed with 0, namely α_0, β_0, ϵ_0, $\boldsymbol{m}_0$, ν_0, and $\boldsymbol{W}_0$. These parameters are so-called prior parameters of the VI, which can be used to influence the behavior of the algorithm in a desired way, e.g., to avoid singularities and to cope with sparse data. The values of these parameters represent prior knowledge that can be set depending on the data set or on the specific application.

The VI algorithm is also able to estimate an appropriate number of components for a dataset. The "effective" number of samples N_j for which a component is responsible can be used as a decision criterion. The higher this number, the more "valuable" is the respective component. If a component is not valuable enough (a test criterion is realized with a threshold), it is simply deleted from the model. That is, the VI training approach must be started with a number of components that must be higher than the number that is expected to be required. The training is performed until a given stopping criterion is met (e.g.,

no or only slight improvements of the likelihood or a fixed number of steps). Point estimates of the model parameters are then obtained by calculating the expected value of the trained distributions.

At this point, we have found parameter estimates for the rule premises ($p(j|\mathbf{x})$, cf. Eq. (1)) in an *unsupervised* manner. Now, we still need the parameters $\xi_{j,c}$ of the rule conclusions $p(c|j)$. These can be obtained in a second, *supervised* step. With I_c we denote the index set of all samples from the overall training set $\mathbf{X}$ for which c is the assigned target class. Then, with $p(j|c) = \int_{\mathbf{x}\in\mathcal{R}_c} p(j|\mathbf{x})\mathrm{d}\mathbf{x}$ where $\mathcal{R}_c$ is the region of the input space associated with class c, we get the maximum likelihood estimates

$$\xi_{j,c} = \frac{1}{N_j} \sum_{n\in I_c} \gamma_{n,j}. \tag{11}$$

This supervised step can also be realized in a slightly different way if labeled data are not available: After the unsupervised step, the components (i.e., rule premises) may be labeled by a human domain expert.

3.3 Knowledge Adaptation Based on Novelty-Awareness

In a changing environment agents must be able to detect the need for generating new rules and to handle this situation appropriately. In the case of an IDA, new knowledge is required if events are observed that are not covered by the current set of classification rules. This can either be due to malicious actions such as an hitherto unknown attack taking place or legitimate actions that deviate from the learned profile (e.g., a newly installed application).

First, we will describe our novel approach for detecting the need for new knowledge (i.e., novelty detection). Here, we only use the continuous part $\mathbf{x}^{\text{cont}}$ as an indicator for novelty. The key measure for our technique is the Mahalanobis distance Δ. We exploit the fact that the squared Mahalanobis distances $\Delta_j^2(\mathbf{x}^{\text{cont}}, \boldsymbol{\mu}_j)$ of samples $\mathbf{x}^{\text{cont}}$ generated by a Gaussian component j to the corresponding center $\boldsymbol{\mu}_j$ are approximately χ^2-distributed with D_{cont} degrees of freedom. Knowing the distribution of the Mahalanobis distances, we can define a hyper-ellipsoid around each center $\boldsymbol{\mu}_j$ such that we can expect that a certain percentage κ of the samples produced by the process which is modeled by component j lies within that hyper-ellipsoid. The radius ρ of these hyper-ellipsoids can be determined by means of the inverted cumulative χ^2 distribution. Based on these hyper-ellipsoids we define a novelty status s_{nov} of the overall classifier that can be regarded as the degree of "satisfaction" with respect to the currently observed situation. This status is updated with every new observation. It is rewarded if the observation is inside the hyper-ellipsoid and penalized otherwise. If the ratio of penalty to reward is equal to the ratio of "inside samples" to "outside samples", there is an equilibrium of penalties and rewards and s_{nov} oscillates around its initial value. In this case, our classifier fits the observed data. If, however, we observe more than $1-\kappa$ percent outside samples (i.e., due to a new process that is not yet covered by the classifier such as a new kind

of attack), s_{nov} is penalized more often and, thus, decreases. If it falls below a user-defined threshold τ, novelty is detected.

To compensate the effect of overlapping components we additionally scale the rewards and penalties with the component responsibilities (cf. Eq. 10). Thus, our novelty detection algorithm works as follows:

Novelty Detection:

1. *Set the percentage κ of samples that are expected to be inside the hyper-ellipsoid (e.g., $\kappa := 0.9$) and penalty ν_{pen} and reward ν_{rew} values with the correct ratio (e.g., $\nu_{\text{rew}} := 0.1$, $\nu_{\text{pen}} := \frac{\kappa}{1-\kappa} \cdot \nu_{\text{rew}}$).*
2. *Determine the set J_{in} of rules for which the sample $\mathbf{x}^{cont}$ is inside the hyper-ellipsoid and the set J_{out} of the remaining rules by comparing the squared Mahalanobis distance of $\mathbf{x}^{cont}$ to the centers $\boldsymbol{\mu}_j$ to a threshold ρ:*

$$J_{\text{in}} := \{j | \Delta_j^2(\mathbf{x}^{cont}, \boldsymbol{\mu}_j) \leq \rho\} \qquad J_{\text{out}} := \{j | \Delta_j^2(\mathbf{x}^{cont}, \boldsymbol{\mu}_j) > \rho\}.$$

   The threshold ρ is obtained by evaluating the inverse cumulative χ^2-distribution for the value κ.
3. *Compute an update value for the overall novelty status of the classifier by scaling the rewards and penalties with the corresponding responsibilities:*

$$\Delta_{\text{nov}}(\mathbf{x}^{cont}) := \eta \cdot \left(\sum_{j \in J_{\text{in}}} \gamma_{n,j} \cdot \nu_{\text{rew}} - \sum_{j \in J_{\text{out}}} \gamma_{n,j} \cdot \nu_{\text{pen}} \right)$$

   with η being the step size controlling the reaction time.
4. *The new novelty status is then*

$$s_{\text{nov}} := s_{\text{nov}} + \Delta_{\text{nov}}(\mathbf{x}^{cont})$$

   where s_{nov} must be initialized appropriately (e.g., with $s_{\text{nov}} := 1$).
5. *If s_{nov} sinks below a given threshold τ (e.g., $\tau := 0.2$), there is a need to integrate one or several new rules into the classifier.*

The algorithm can be parametrized to show different behavior, i.e., if new processes are expected to emerge distant to existing ones and the detection delay should be short, larger values for κ (e.g., $\kappa = 0.95$) should be used. A more deliberate behavior can be achieved with smaller values ($\kappa = 0.7$). It is also possible to use multiple instances with different parameterizations in parallel.

Whenever novelty is stated, the rule set must be adapted accordingly by adding new rules. Basically, we use the VI technique on a sliding window of recent samples to find new rule premises. To avoid changes of the already existing premises, the centers and covariance matrices of existing components are fixed and only those of new components are adapted. The mixture coefficients can either be re-estimated based on the sliding window or set to identical values. A rule conclusion, i.e., an estimate of the parameters of the distribution $p(c|j)$ for a new component j, can then be obtained in various ways:

1. Application experts (e.g., system administrators) can be asked to label a set of recently observed samples (e.g., measured within a sliding window). These labels are then used to determine values for the parameters $\xi_{j,c}$.
2. Application experts can be asked to label a new rule j, i.e., to assign it uniquely to one of the classes.
3. In the case of rule exchange between IDA, certain rules, and in particular their conclusions, may be taken over from other IDA.

Altogether, we can state that the adaptation of rule premises can be done in an unsupervised way, i.e., autonomously by the agents themselves. For the rule conclusions we need application experts in some application scenarios, but their effort can be kept as low as possible if rules are taken over from other agents whenever possible and experts are asked to label rules instead of a (often large) number of samples which can be done much more efficiently.

4 Experiments

To analyze the performance of our proposed techniques we use parts of the well-known DARPA intrusion detection evaluation data set [16] that consists of several weeks of labeled network data (i.e., legitimate normal network traffic interleaved with various attacks) which was generated within a simulated environment. The network architecture as well as the generated network traffic have been designed to be similar to that of an Air Force base. We are aware of the various critique of the DARPA data (e.g. [17]). In order to achieve fair and realistic results, we carefully analyzed all known deficiencies and omitted features that could bias detector performance.

We used the TCP/IP network dump as input data for an agent. At the sensor layer TCP connections are reassembled and statistical information (i.e., events) are extracted and handed over to the detection layer. Each event consists of features that are typically used for intrusion detection, i.e., two categorical dimensions (source and destination port) and ten continuous dimensions (e.g., duration, number of transmitted packets, entropy of the exchanged information). We removed all attack connections from the first week of the DARPA data set and used the result to train an initial detection model for the agent that describes the expected normal network traffic. The agent is equipped with two instances of our proposed novelty detection technique (one parametrized with $\kappa = 0.97, \eta = 0.05$ to detect distant and one with $\kappa = 0.80, \eta = 0.1$ to detect close new processes) and it is able to perform self-adaptation. When a new rule premise is generated, a simulated human expert is asked to provide a conclusion (which is calculated here using the labeled connections). We confront the agent with weeks 2 to 5 (attacks with less than 20 connections were left out). After every week, we reset the agent to its initial model to prevent influences between the weeks.

Table 1 (left) shows the classification rate (CR), missing alert rate (MA), false alert rate (FA), the total number of connections (i.e., events), and the number of newly generated rules for week 2. First, note that obviously the normal traffic of week 2 differs from week 1 as two additional rules are generated. The attacks

back and *ipsweep* are successfully detected with good classification rates. *Guest* and *portsweep* only consist of a few connections and, thus, the delay until a new rule is generated significantly reduces the classification rate. Interestingly, *ipsweep* shows a high rate of false alerts. A closer inspection of the misclassified connections showed, however, that these connections are suspicious (i.e., they are directed at closed ports and consist only of two packets) and should definitely be reported to a system administrator as they indicate an erroneous program configuration.

Table 1 (right) shows the results for week 3. The normal traffic significantly differs from week 1 as the agent learned 18 new rules. All attacks are successfully detected. Again, *portsweep* being a short attack results in a lower classification rate due to the detection delay.

Table 1. Classification Results for Week 2 (Left) and Week 3 (Right)

Type	CR	MA	FA	events	rules
Normal	n/a	n/a	n/a	182 932	2
Back	95.5%	4.6%	1.4%	983	3
Guest	74.0%	26.0%	0.0%	50	1
Ipsweep	89.9%	10.1%	72.1%	855	2
Portsweep	63.6%	36.4%	8.3%	99	4

Type	CR	MA	FA	events	rules
Normal	n/a	n/a	n/a	54 893	18
Back	92.1%	7.9%	0.3%	999	3
Neptune	100%	0.0%	0.4%	185 652	1
Nmap	95.9%	4.1%	9.1%	941	1
Portsweep	70.0%	30.0%	5.6%	100	4

Table 2. Classification Results for Week 4 (Left) and Week 5 (Right)

Type	CR	MA	FA	events	rules
Normal	n/a	n/a	n/a	49 159	10
Neptune	98.9%	1.1%	0.2%	798	1
Portsweep	98.4%	1.6%	5.7%	1 971	3
Satan	16.7%	83.3%	2.4%	4 003	2
Warezclient	72.8%	27.2%	0.2%	419	2

Type	CR	MA	FA	events	rules
Normal	n/a	n/a	n/a	44 130	11
Neptune	100%	0.0%	0.4%	419 832	1
Portsweep	96.4%	3.6%	2.9%	2 238	7
Satan	5.8%	94.1%	1.9%	204	1

The results for week 4 are outlined in Table 2 (left). Obviously, the attack *satan* is very hard to detect for our agent. A closer inspection showed, that the first of the 10 newly generated rules for normal traffic covers a major part of all *satan* connections. We analyzed the events that resulted in the generation of this rule and, again, found a number of unsuccessful connection attempts. However, as they are labeled as being normal connections our simulated human expert provides a "normal" conclusion for the rule. A real human expert would certainly provide a different conclusion for these suspicious connections.

Neptune and *portsweep* are very well detected in week 5 (cf. Table 2, right). Again, such as in week 4, one of the new rules for normal traffic covers most of the *satan* connections.

These experiments showed very promising results of our proposed techniques. Our initial model was trained using only normal traffic but for all attacks corresponding rule premises were generated and for nearly all attacks good classification results were obtained. The classification rates of the *satan* attack suffered from a number of unsuccessful connection attempts contained in the normal traffic that led to the creation of corresponding normal rules. Thus, these misclassifications can be regarded as an artifact caused by our expert simulation.

5 Conclusion and Outlook

In this work we laid the methodological basis for IDA that recognize novel kinds of attacks, react accordingly by creating new rules, and (in the future) collaborate by exchanging these locally learned rules. The self-adaptation of the IDA is performed in a very efficient manner that reduces the need for a human application expert to a minimum: System administrators are confronted with rule premises for which appropriate conclusions must be found. Thus, we try to avoid situations where system administrators must analyze huge amount of alerts to build new rules. We investigated how these techniques perform on some well-known benchmark intrusion data. In the future, we will combine the classifiers presented here with conventional intrusion detection systems such as Snort [18] to improve the classification rates and we will develop DIDS based on OC principles. We will also consider categorical input attributes of a classifier in our novelty-awareness techniques (detection and reaction) and improve the temporal behavior of these techniques.

It is obvious that the proposed novelty-awareness techniques may also be used in other applications to support intelligent technical systems (e.g., in the field of OC) in their task to resolve problems with knowledge deficits in a (partly or completely) self-organizing way. Thus, novelty detection and reaction techniques will become a fundamental OC principle. We will use the techniques for knowledge exchange in intelligent distributed systems, too. This kind of collective intelligence is biologically inspired in the sense that these systems follow the human archetype: Humans not only learn by exchanging information (e.g., observed facts) but also by teaching each other learned knowledge (e.g., rules) and experience gained with the application of this knowledge.

Acknowledgment

This work was supported by the German Research Foundation (DFG) under grants SI 674/3-2 and SI 674/3-3 (priority program Organic Computing).

References

1. Müller-Schloer, C.: Organic computing – on the feasibility of controlled emergence. In: IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2004), Stockholm, Sweden, pp. 2–5 (2004)
2. Würtz, R.P. (ed.): Organic Computing. Understanding Complex Systems. Springer, Heidelberg (2008)
3. Buchtala, O., Grass, W., Hofmann, A., Sick, B.: A fusion-based intrusion detection architecture with organic behavior. In: The first CRIS Int. Workshop on Critical Information Infrastructures (CIIW), Linköping, pp. 47–56 (2005)
4. Fisch, D., Hofmann, A., Hornik, V., Dedinski, I., Sick, B.: A framework for large-scale simulation of collaborative intrusion detection. In: IEEE Conf. on Soft Computing in Industrial Applications (SMCia/ 2008), Muroran, Japan, pp. 125–130 (2008)

5. Hofmann, A., Sick, B.: On-line intrusion alert aggregation with generative data stream modeling. IEEE Tr. on Dependable and Secure Computing (2010) (status: accepted), http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.36
6. Axelsson, S.: Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers University of Technology, Department of Computer Engineering (2000)
7. Snapp, S.R., Brentano, J., Dias, G.V., Goan, T.L., Heberlein, L.T., Ho, C.L., Levitt, K.N., Mukherjee, B., Smaha, S.E., Grance, T., Teal, D.M., Mansur, D.: DIDS (distributed intrusion detection system) – motivation, architecture, and an early prototype. In: Proc. of the 15th IEEE National Computer Security Conf., Baltimore, MD, pp. 167–176 (1992)
8. Yegneswaran, V., Barford, P., Jha, S.: Global intrusion detection in the domino overlay system. In: Proc. of the Network and Distributed System Security Symp., NDSS 2004, San Diego, CA (2004)
9. Chatzigiannakis, V., Androulidakis, G., Grammatikou, M., Maglaris, B.: A distributed intrusion detection prototype using security agents. In: Proc. of the 6th Int. Conf., on Software Engineering, Artificial Intelligence, Networking and Parallel and Distributed Computing, Beijing, China, pp. 238–245 (2004)
10. Zhang, Y.F., Xiong, Z.Y., Wang, X.Q.: Distributed intrusion detection based on clustering. In: Proc. of 2005 Int. Conf. on Machine Learning and Cybernetics, Guangzhou, China, vol. 4, pp. 2379–2383 (2005)
11. Dickerson, J.E., Juslin, J., Koukousoula, O., Dickerson, J.A.: Fuzzy intrusion detection. In: Proc. IFSA World Congress and 20th North American Fuzzy Information Processing Society (NAFIPS) Int. Conf., Vancouver, BC, pp. 1506–1510 (2001)
12. Kim, J., Bentley, P.: The artificial immune model for network intrusion detection. In: 7th European Conf. on Intelligent Techniques and Soft Computing (EUFIT 1999), Aachen, Germany (1999)
13. Folino, G., Pizzuti, C., Spezzano, G.: Gp ensemble for distributed intrusion detection systems. In: Proc. of the 3rd Int. Conf. on Advances in Pattern Recognition, Bath, U.K, pp. 54–62 (2005)
14. Fisch, D., Sick, B.: Training of radial basis function classifiers with resilient propagation and variational Bayesian inference. In: Proc. of the Int. Joint Conf. on Neural Networks (IJCNN 2009), Atlanta, GA, pp. 838–847 (2009)
15. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)
16. Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., Zissman, M.A.: Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In: DARPA Information Survivability Conf. and Exposition (DISCEX), Hilton Head, SC, vol. 2, pp. 12–26 (2000)
17. McHugh, J.: Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. ACM Tr. on Information and System Security 3(4), 262–294 (2000)
18. Roesch, M.: Snort – lightweight intrusion detection for networks. In: LISA 1999: Proc. of the 13th USENIX Conf. on System Administration, Berkeley, CA, pp. 229–238 (1999)