Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems

Martin Rehák[1], Eugen Staab[2], Volker Fusenig[2], Michal Pěchouček[1], Martin Grill[3,1], Jan Stiborek[1], Karel Bartoš[3,1], and Thomas Engel[2]

[1] Department of Cybernetics, Czech Technical University in Prague
{rehak,pechoucek,grill,stiborek,bartos}@agents.felk.cvut.cz
[2] Faculty of Science, Technology and Communication, University of Luxembourg
{eugen.staab,volker.fusenig}@uni.lu
[3] CESNET, z. s. p. o., Prague, Czech Republic

Abstract. Our work proposes a generic architecture for runtime monitoring and optimization of IDS based on the challenge insertion. The challenges, known instances of malicious or legitimate behavior, are inserted into the network traffic represented by NetFlow records, processed with the current traffic and the system's response to the challenges is used to determine its effectiveness and to fine-tune its parameters. The insertion of challenges is based on the threat models expressed as attack trees with attached risk/loss values. The use of threat model allows the system to measure the expected undetected loss and to improve its performance with respect to the relevant threats, as we have verified in the experiments performed on live network traffic.

1 Introduction

One of the principal problems of the intrusion detection systems based on the anomaly detection [1] principles is their error rate, both in terms of false negatives (undetected attacks) and false positives, i.e. legitimate traffic labeled as malicious. This problem is amplified by the fact that the sensitivity (and consequently the error rate) varies dynamically as a function of the background traffic. For example, an attack that would be easily discovered in the lower nighttime traffic will pass undetected during the day, on the system with identical settings. In this work, we address the problem of correct IDS monitoring and dynamic reconfiguration, in order to provide the operators with:

- an estimate of system sensitivity/error rate, given the current network traffic and a threat model, and
- autonomous system reconfiguration, based on the system monitoring and the threat model.

In order to perform these tasks, we use the concept of challenges [2] (or fault injection) from the field of autonomic computing, which allows us to measure the response of the system with respect to a small subset of *challenges*, known instances of malicious or legitimate behavior, inserted into the traffic observed on the network. The response of the system and its individual components to the inserted challenges is used to determine

E. Kirda, S. Jha, and D. Balzarotti (Eds.): RAID 2009, LNCS 5758, pp. 61–80, 2009.

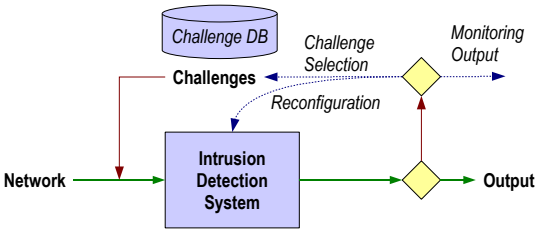


Fig. 1. Adaptation process overview

its current error rate in terms of estimated ratio of false positives/false negatives (see Fig. 1). It is also used to adapt the system behavior and to select and/or create optimal system settings.

This generic concept is verified by its integration with the CAMNEP intrusion detection system [3][4], which is based on a multi-stage combination of several network behavior analysis algorithms processing the NetFlow [5] data. In Section 2, we briefly discuss the relevant properties of the CAMNEP system, which was augmented with the processes described in this paper. Then, we present the self-adaptive architecture integrated with the underlying system and discuss the crucial elements of the architecture (Section 3), such as dynamic classifier selection and optimization of number of challenges and their composition. These sections describe the core contribution of this work.

2 CAMNEP System

The self-optimization techniques presented in this paper were integrated with the CAMNEP network intrusion detection system [3], based on the Network Behavior Analysis (NBA) approach [6]. This system processes NetFlow/IPFIX data provided by routers or other network equipment and uses this information to identify malicious traffic by means of collaborative, multi-algorithm anomaly detection. The system uses the multi-algorithm and multi-stage approach to optimize the error rate, while not compromising the performance of the system. The self-monitoring and self-adaptation techniques are very relevant in this context, as they allow to improve the error rate with only a minimal and controllable impact on its efficiency.

The NetFlow network traffic data is structured in records, and each record describes one *flow*. A flow can be described as an unidirectional component of TCP connection (or its UDP/ICMP equivalent) and contains all packets with the same source IP, destination IP, source and destination port and protocol (TCP/UDP/ICMP). A flow record contains this basic information, as well as other information, such as the number of packets/bytes transferred, duration and TCP flags encountered in the packets of the flow. The flow records are aggregated over a predefined observation period (typically 1-5 minutes). When the observation period elapses, the data is read out for analysis, and a new observation period begins.

The system contains two principal classes of classifying agents, which are able to evaluate the received traffic:

Detection agents (agents A and B in Fig. 2) analyze raw network flows by their anomaly detection algorithms, exchange the anomalies between them and use the aggregated anomalies to build and update the long-term anomaly associated with the abstract traffic classes built by each agent. These traffic classes describe various behaviors, as they can be distinguished based on the features used by the anomaly detection methods integrated into the system. Each detection agent uses one of the five anomaly detection methods listed herein. Each of the methods works with a different traffic model based on a specific combination of aggregate traffic features, such as: (*i*) entropies of flow characteristics for individual source IP addresses [7], (*ii*) deviation of flow entropies from the PCA-based prediction model of individual sources [8], (*iii*) deviation of traffic volumes from the PCA-based prediction for individual major sources [9], (*iv*) rapid surges in the number of flows with given characteristics from the individual sources [10] and (*v*) ratios between the number of destination addresses and port numbers for individual sources [11].

All detection agents map the same flows, together with the shared evaluation of these events, the aggregated immediate anomaly of these events determined by their anomaly detection algorithms, into the traffic clusters built using different features/metrics, thus building the aggregate anomaly hypothesis based on different premises. The *aggregated anomalies* associated with the individual traffic classes are built and maintained using the classic trust modeling techniques (not to be confused with the way trust is used in this work). The detection agents evaluate the anomaly of each network flow on the whole [0,1] interval, and the output of the detection agents is integrated by the aggregation agents.

Aggregation agents α_1 from the set $A = \{\alpha_1, \ldots, \alpha_g\}$ represent the various aggregation operators used to build the joint conclusion regarding the normality/anomaly of the flows from the individual opinions provided by the detection agents. Each agent uses a distinct averaging operator (based on order-weighted averaging [12] or simple weighted averaging) to perform the $R^{g_{det}} \rightarrow R$ transformation from the g_{det}-dimensional space to a single real value, thus defining one composite system output that integrates the results of several detection agents. The aggregation agents also dynamically determine the threshold values used to transform the continuous aggregated anomaly value in the $[0, 1]$ interval into the crisp normal/anomalous assessment for each flow. The value of the threshold is either relative (i.e. leftmost part of the distribution) or absolute, based on the evaluation of the agent's response to challenges.

The detection and aggregation agents annotate the individual flows φ with a continuous *anomaly/normality* value in the $[0, 1]$ interval, with the value 1 corresponding to perfectly normal events and the value 0 to completely anomalous ones. This continuous anomaly value describes an agent's opinion regarding the anomaly of the event, and the agents apply adaptive or predefined thresholds to split the $[0, 1]$ interval into the normal and anomalous classes. The threshold applied (and dynamically maintained) by the aggregation agents divides the flows into two classes: *normal* and *anomalous*. The anomalous flows are those whose anomaly falls below the threshold, while the normal

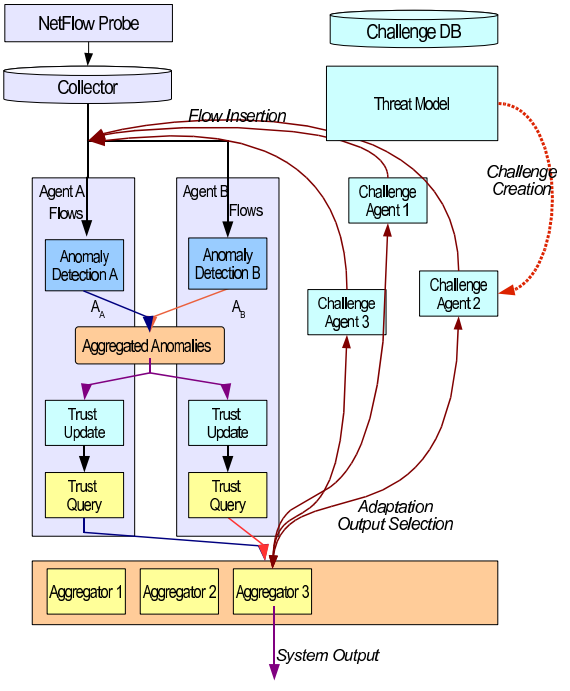


Fig. 2. Adaptation process in the CAMNEP system

flows are those, whose anomaly is above the threshold. This distinction allows us to introduce the components of the error rate. *False Positives* (FP) are the legitimate flows classified as anomalous, while the *False Negatives* (FN) are the malicious flows classified as normal. Most standalone NBA methods suffer from a very high rate of false positives, which makes them unpractical for deployment. The static multi-stage process of the original CAMNEP system already removes a large part of false positives, while not increasing the rate of false negatives, and the goal of the self-optimization techniques is to further improve the effectiveness of the system.

3 IDS Monitoring Architecture

The monitoring and adaptation components of the CAMNEP system implement the high-level functional schema introduced in Fig. 1. The reconfiguration action (as shown in Fig. 1) is the identification of the optimal anomaly aggregation function that achieves the best separation between the legitimate and malicious challenges. Assuming that these challenges are representative of the traffic in the network and the expected attacks, such aggregation should also optimize the performance against the actual threats in the current network traffic. The adaptation process also provides the user with the estimates of system detection effectiveness against the threats defined in the threat model, as it presents the effectiveness values for the currently selected aggregation function.

The background traffic is one of the adaptation process indirect inputs, as it influences the performance of the individual anomaly detection algorithms. As the network traffic is highly unpredictable, it is very difficult to predict which aggregation function will be chosen, especially given the fact that the challenges are selected from the DB using a stochastic process with a pseudo-random generator unknown to a potential attacker. The attacker therefore faces a dynamic IDS system that unpredictably switches its detection profile between several different profiles with utility (i.e. detection performance) values close to the optimum, and has to operate in a manner which would evade any of these profiles. This unpredictability, together with the additional robustness achieved by the use of multiple algorithms, makes the IDS evasion a much more difficult task than simply avoiding a single intrusion detection method[13].

The self-adaptation process (detailed in Fig. 2) is based on the insertion of challenges into the background of network flow data observed by the system. The challenges are represented as sets of NetFlow records, corresponding to classified incidents observed in the past. These records are generated by short lived, challenge specific *challenge agents* and are mixed with the background traffic, so that they cannot be distinguished from the background by the detection/aggregation agents. They are processed together with the rest of the traffic, used to update the anomaly detection mechanism data and trust models of individual detection agents and are evaluated with the rest of the traffic. Once the processing is completed, the challenge flows are re-identified by their respective challenge agents, removed from the user output and the anomaly attributed to these flows by individual aggregation agents is used to evaluate these agents and to select the optimal output agent for the current network conditions.

There are two broad types of challenges. The *malicious challenges* correspond to known attack types, while the *legitimate challenges* represent known instances of legitimate events that tend to be misclassified as anomalous. We further divide the malicious challenges into broad classes (denoted $AC_1, \ldots, AC_k, \ldots$) characterized by the type of the attack, such as fingerprinting/vertical scan, horizontal scan, password brute forcing, etc. These classes are used to make the connection between the threat models in Section 4.1 and the challenge selection. With respect to each of these attack classes, we characterize each aggregation agent by a probability distribution, empirically estimated from the continuous anomaly values attributed to the challenges from this class, as we can see in Fig. 3. We also define a single additional distribution for all legitimate challenges.

We assume that the anomaly values of both the legitimate and all types of malicious challenges define normal distributions, with the parameters $\bar{x}^k$ and σ_x^k for the k-th class AC_k of malicious challenges and $\bar{y}$ and σ_y for the legitimate ones[1]. The distance between the estimated mean values of both distributions ($\bar{x}^k$ and $\bar{y}$), normalized with respect to the values σ_x^k and σ_y represents the quality of the aggregation agent with respect to a given attack class. The *effectiveness* of the agent, defined as an ability to distinguish between the legitimate events and the attacks is defined as a weighted average of the effectiveness with respect to individual classes and will be estimated by the

[1] Normality of both distributions is not difficult to achieve, provided that the attack classes are properly defined and that the challenge samples in these classes are well selected, i.e. comparable in terms of size and other parameters.

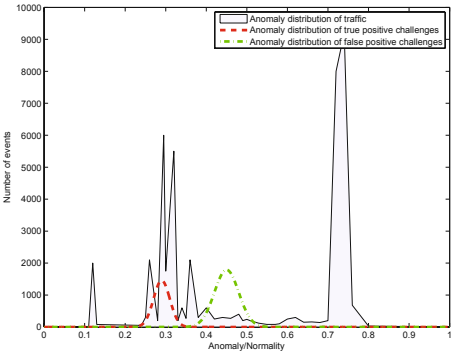


Fig. 3. Distribution of challenges on the background of the anomalies attributed to the traffic from one traffic observation interval. The distribution of anomaly of the malicious challenges (from one class) is on the left side of the graph, while the legitimate events are on the right.

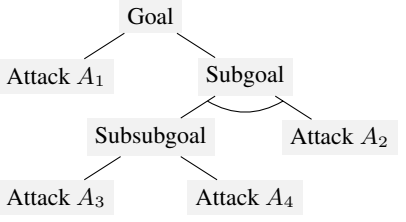


Fig. 4. Example Structure of an Attack Tree

trust modeling approach introduced in Sect. 5. In order to perform the above-described self-adaptation process, we need to address three important issues:

- offline selection of appropriate challenges and estimation of their relative importance (Sections 4 and 4.3),
- dynamic selection of the optimal aggregation agent to be used as a system output (Section 5), and
- dynamic determination of the optimal number of challenges.

4 Threat-Based Approach to Challenge Selection

In this section, we will present a method for challenge selection based on explicit threat modeling. We define a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of relevant *threats* as identified by the network administrator. Each threat is described by an attack tree, which specifies the adversary's attacks necessary to realize the threat. For each threat T_i, the system administrator has specified the expected damage $D(T_i)$, which would be caused should the attacker realize the threat. Our system uses challenges to evaluate its internal components in terms of accuracy and selects the most accurate component. Each challenge tests for a specific class of attacks. Therefore, the detection of threats can be directed by prioritizing those challenges that test for the most damaging threats.

In the following, we shortly review the concept of *attack trees* (Sect. 4.1) and show how they can be formulated in propositional calculus (Sect. 4.2). The latter allows us to minimize attack trees, and so bring them into an expedient form for further processing. We use the minimized attack trees to determine the composition of challenges for evaluating the internal components (Sect. 4.3).

4.1 Attack Trees

Attack trees depict how an attacker can attain a certain goal, e.g., to gain unauthorized access to a system resource. This overall goal constitutes a threat to a security system and builds the root of an attack tree.

The attack tree shows the alternative ways of how an attacker can reach the root, and so realize the threat. As formalized in [14], an attack tree is composed of AND and OR branches. Figure 4 shows a simple example of an attack tree structure. In this figure, the branch with a connectional arc depicts an *AND branch*, all other branches are *OR branches*. To reach the root, the attacker has to conduct a series of basic network attacks, e.g., "horizontal scan", which we call the *atomic attacks*. These atomic attacks constitute the leafs of an attack tree. An attacker "reaches" a leaf if he conducts the corresponding attack. Then, either if all children of a node with an AND branch are reached, then the node itself is reached. Similarly, an OR branched node is reached, if at least one of its children is reached. This way, starting at the leafs by conducting atomic attacks, an attacker can work its way up to the root. For our example in Fig. 4, the attacker can for instance reach the root by performing the attacks A_3 and A_2.

The principal advantages of the attack tree formalism are its simplicity, relatively high expressivity, and generality: an attack tree-level description of the threat is easily transferable between the networks and can be thus reused.

4.2 Attack Trees in Propositional Logic

We can say, an attacker can reach the root node by reaching specific subsets of the leafs. In this section we show how these specific subsets can be identified and minimized in a neat manner. First, we represent an attack tree in propositional logic. A formula corresponding to a tree should become true *iff* the main goal in the attack tree is attained. To build such a formula, we first create a literal for each atomic attack. Now, we successively go through the tree (starting from the root node), and connect all children of a node by the appropriate logic operation (OR for disjunctive branches, AND for conjunctive branches). Parentheses are used to group the children together. For the example tree shown in Fig. 4 this results in the formula:

$$(A_1) \vee ((A_3 \vee A_4) \wedge (A_2)) \; . \tag{1}$$

A formula is in Disjunctive Normal Form (DNF) iff it is a disjunction of conjunctive clauses. A formula is canonical, if all clauses contain all variables. We can bring any formula into canonical DNF by building a truth table that contains all variables, and taking all rows that evaluate to *true* as clauses. For our toy example in Fig. 4 that would result in:

$$(A_1 \wedge A_2 \wedge A_3 \wedge A_4) \tag{2}$$
$$\vee(A_1 \wedge A_2 \wedge A_3 \wedge \neg A_4) \tag{3}$$
$$\vee(A_1 \wedge A_2 \wedge \neg A_3 \wedge A_4) \tag{4}$$
$$\vee \quad \ldots \tag{5}$$

Having an attack tree in canonical DNF, we can say, that an attacker realizes the threat if he succeeds to make at least one clause true. However, there is still much redundancy in the formula. For example, lines 2 and 3 together are logically equivalent to $A_1 \wedge A_2 \wedge A_3$. To remove all redundancy from the formula, we simply apply the Quine-McCluskey algorithm [15]. Note that when simplifying attack tree formulas, clauses will only contain positive literals. For the attack tree in Fig. 4, we finally get:

$$(A_1) \vee (A_3 \wedge A_2) \vee (A_4 \wedge A_2) \; . \tag{6}$$

A formula in DNF can be written as a set of clauses $\{C_1, C_2, \dots\}$ where each clause C_i is a set of positive literals $\{l_{i1}, l_{i2}, \dots\}$. We will write $F(T)$ for the minimal formula in DNF that corresponds to attack tree T. The attack tree from Fig. 4 can be formalized as:

$$F(T) = \{\{A_1\}, \{A_2, A_3\}, \{A_2, A_4\}\} \; . \tag{7}$$

4.3 Attack Tree Valuation

In this section, we first show how different attack classes can be prioritized, depending on the expected damage of the successful attacks, i.e. the attack tree root being attained by the adversary. We then show how the resulting priorities can be used to determine the composition of challenges for adapting the IDS. Finally, we exemplify the procedure with an example for a specific attack tree.

We assume, that a set of n detectable attacks $\mathcal{A} = \{A_1, \dots, A_n\}$ and general network conditions are known to the configured IDS. These attacks are classified into K attack classes $\{AC_1, \dots, AC_K\}$, with $\bigcup_k AC_k = \mathcal{A}$. We don't require that all attacks in an attack class are known, as the system is able to assess its effectiveness against the attacks inserted into the traffic in real-time. However, we require a sufficient set of attacks for each attack class, in order to use these samples as challenges.

The problem now is to prioritize the detection of attack classes. To this end, the following criteria should be fulfilled:

Attack trees: An attacker has a certain goal (which determines the attack tree T). Attack trees that cause more damage should be prioritized.

Clauses: An attacker tries to make one clause true in a chosen formula $F(T)$. Any clause made true causes the same damage $D(T)$. So each clause is assigned the same priority.

Literals: For making a chosen clause true, an attacker needs to make true *all* literals in this clause to cause damage $D(T)$. Therefore, all literals belonging to the same clause should be equally prioritized.

To fulfill the last two criteria, we compute the priority of an attack A_i within a tree T_j as follows:

$$P(A_i, T_j) := \frac{1}{|F(T_j)|} \sum_{\substack{C_k \in F(T_j), \\ \text{with } A_i \in C_k}} \frac{1}{|C_k|} \; . \tag{8}$$

The reader can easily verify that if A_i is not in T_j, then its priority within the tree is zero. Also, the sum of all priorities of the attacks in the tree is 1. To fulfill the first criterion, we additionally weight each tree T_j according to the damage $D(T_j)$ and get the final priority for an attack A_i by summing over all attack trees:

$$P(A_i) := \frac{1}{\sum_{T_j \in \mathcal{T}} D(T_j)} \cdot \sum_{T_k \in \mathcal{T}} D(T_k) \cdot P(A_i, T_k) \; . \tag{9}$$

Because of the normalization, again the priorities of all attacks sum up to 1. Hence, we can use these priorities to directly determine the ratio of challenges to test the respective attacks.

Procedure. In order to calculate the priorities of the attacks in $\mathcal{A}$, we propose the following procedure:

1. For each tree $T_i \in \mathcal{T}$ do:
   (a) Prune all impossible and non-detectable attacks from the tree.
   (b) Build $F(T_i)$: Transform the tree into a logical formula, bring it into DNF and minimize it (as in Sect. 4.2).
2. Compute $P(A_i)$ for each attack A_i as shown in formula (9).
3. For each attack class AC, add the priorities for all attacks in that class:

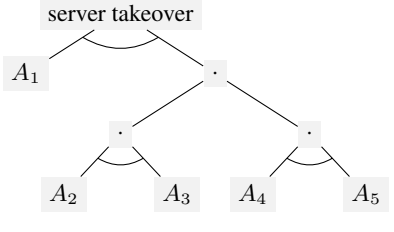
$$P(AC) = \sum_{A_i \in AC} P(A_i) \; . \tag{10}$$

   The ratio $P(AC)$ is a proportion of challenges from the class AC, and we will use it to as a weight in Eq. 22.

Example. In this section we show how the priorities are computed for a set of two very simple example attack trees T_1 and T_2 shown in Fig. 5 and 6 respectively. We estimate the damages of the trees to be $D(T_1) = 900$ and $D(T_2) = 100$. The minimal formulas in DNF for the two attack trees are:

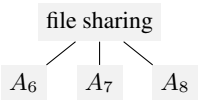
$$F(T_1) = \{\{A_1, A_2, A_3\}, \{A_1, A_4, A_5\}\} \; , \tag{11}$$

$$F(T_2) = \{\{A_6\}, \{A_7\}, \{A_8\}\} \; . \tag{12}$$



attack description		*attack class*
A_1	horizontal scan	AC_1
A_2	fingerprinting	AC_2
A_3	buffer overflow	AC_3
A_4	SSH brute force request	AC_4
A_5	SSH brute force response	AC_4

Fig. 5. Example Attack Tree T_1



attack	*description*	*attack class*
A_6	download	AC_5
A_7	upload	AC_5
A_8	directory node	AC_5

Fig. 6. Example Attack Tree T_2

We can now compute $P(A_i, T_j)$ for all attacks. Clearly $P(A_1, T_2) = 0$, so let us look at $P(A_1, T_1)$:

$$P(A_1, T_1) = \frac{1}{|F(T_1)|}\left(\frac{1}{|C_1|} + \frac{1}{|C_2|}\right) = \frac{1}{2}\left(\frac{1}{3} + \frac{1}{3}\right) = \frac{1}{3}\,. \tag{13}$$

Analogously we obtain:

$$P(A_2, T_1) = P(A_3, T_1) = P(A_4, T_1) = P(A_5, T_1) = \frac{1}{2} * \frac{1}{3} = \frac{1}{6}\,. \tag{14}$$

For attack tree T_2 we get:

$$P(A_6, T_2) = P(A_7, T_2) = P(A_8, T_2) = \frac{1}{3}\,. \tag{15}$$

Now, combining the two trees according to their expected damage, we obtain:

$$P(A_1) = \frac{D(T_1)}{D(T_1) + D(T_2)} \cdot P(A_1, T_1) = \frac{9}{10} \cdot \frac{1}{3} = \frac{3}{10}\,. \tag{16}$$

In the same way, we obtain for the other attacks:

$$P(A_2) = P(A_3) = P(A_4) = P(A_5) = \frac{3}{20}\,, P(A_6) = P(A_7) = P(A_8) = \frac{1}{30}\,. \tag{17}$$

Finally, we can compute the attack class priorities:

$$P(AC_1) = \frac{3}{10},\ P(AC_2) = P(AC_3) = \frac{3}{20}\,, P(AC_4) = \frac{3}{10},\ P(AC_5) = \frac{1}{10}\,. \tag{18}$$

5 Dynamic Aggregation Agent Selection

The insertion of challenges into the real traffic is not only a difficult problem from the technical perspective (due to the high volume of events processed in near-real-time and hard performance limitations of the system), but can also influence the effectiveness of the aggregation agents based on anomaly detection approaches. As these agents are not able to distinguish the challenges from the real events, the challenges are included in their traffic model, making it less representative of the background traffic and therefore reducing its predictive ability.

In this section, we present a trust-based algorithm which dynamically determines *the best aggregation agent* and also the *optimal number of challenges necessary for the reliable identification of the best aggregation agent*, while taking into account the: (*i*) past effectiveness of the individual aggregation agents and (*ii*) number of aggregation agents and the perceived differences in their effectiveness. We decided to use a trust-based approach for evaluating the aggregation agents, because it not only eliminates the noise in the background traffic and randomness of the challenge selection process, but accounts for the fact that attackers might try to manipulate the system by inserting misleading traffic flows. An attacker could insert fabricated flows [13] hoping they would cause the system to select an aggregation agent that is less sensitive to the threat the attacker actually intends to realize. When using trust, one tries to avoid this manipulation by dynamically adapting to more recent actions of an attacker.

For each time step $i \in \mathbb{N}$, the algorithm proceeds as follows:

1. Let each aggregation agent classify a set of challenges from different attack classes and selected legitimate challenges.
2. Update the *trust value* of each aggregation agent, based on its performance on the challenges in time step i.
3. Accept the output of the aggregation agent with the highest trust value as classification of the remaining events of time step i.

As we have stated above, we challenge detection and aggregation agents in each time step i with the sets of flows for which we already know the actual class, i.e. whether they are malicious or legitimate. So, we challenge an aggregation agent α with a set of malicious events, belonging to K attack classes and a set of legitimate events drawn from a single class. With respect to each class of attacks k, the performance of the agent is described by a mean and a standard deviation: $(\bar{x}^k, \sigma_x^k)$ for the set of malicious challenges and $(\bar{y}, \sigma_y)$ for the set of legitimate challenges. Both means lie in the interval $[0, 1]$, and $\bar{x}^k$ close to 0 and $\bar{y}$ close to 1 signify accurate classifications of the agent respectively (see Fig. 7). Based on this performance in time step i, we define the trust

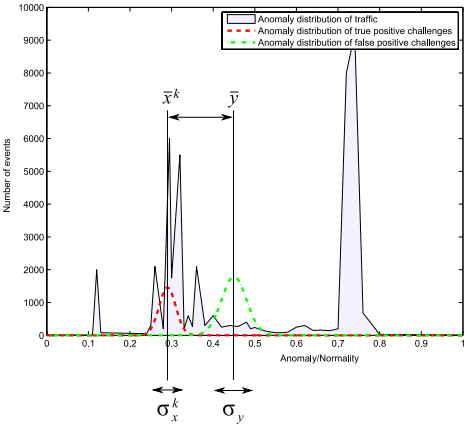


Fig. 7. Performance measures used for computing one trust experience

experience $t_\alpha^{i,k}$ with that aggregation agent α as follows:

$$t_\alpha^{i,k} = \frac{\bar{y} - \bar{x}^k}{\sigma_y + \sigma_x^k} \ . \tag{19}$$

The intention behind this formula is that an agent is more trustworthy, if its classifications are more *accurate* ($\bar{x}^k$ is low and $\bar{y}$ is high), and more *precise* (the standard deviations are low). Note that $t_\alpha^{i,k}$ lies in $(-\infty, \infty)$; however $t_\alpha^{i,k}$ will rarely be negative in practice.

To get the attack-class specific trust value T_α^k for an agent α, we aggregate the past trust experiences with that agent regarding the challenges from class k:

$$T_\alpha^k = \sum_i w_i * t_\alpha^{i,k} \ , \tag{20}$$

where w_i are weights that allow recent experiences a higher impact. This is done because older experiences are expected to be less significant than more recent ones. In our current system, the weights decrease exponentially. The system receives the input events in 5 minute batches, and assigns the same weight to all events in each batch. The weight of the challenges from the batch i is determined as:

$$w_i = \frac{1}{W} e^{(j-i)\frac{\ln(0.1)}{4}} \ , \tag{21}$$

where the j denotes the current time step, and the value of the coefficient $\frac{\ln(0.1)}{4}$ was selected so that challenges from the fifth batch (the oldest one being used) are assigned a weight of 0.1 before the normalization. The normalization is performed simply by dividing all weights by the sum of their un-normalized values W to ensure that $\sum w_i = 1$. We are currently using the challenges from the last 5 batches, meaning that the $(j - i)$ part of the exponent takes the values between 0 and 4. Please note that the specific assignment of weights w_i is highly domain specific, and is only included as an illustration of the general principle.

The final trust value T_α^i for the aggregation agent α is determined as a linear combination of the partial, attack class-specific values T_α^k:

$$T_\alpha^i = \sum_{k=1}^{K} P(AC_k) \cdot T_\alpha^k \ , \tag{22}$$

where the weights $P(AC_k)$ attributed to the trustworthiness of the individual classes are derived from Eq. 10.

5.1 Optimizing Number of Challenges

The number of challenges used as basis for the computation of the trust experiences $t_\alpha^{i,k}$ should be as small as possible while at the same time providing accurate results for the trust experiences. This means that we want to know the minimum number of challenges n for computing $\bar{x}^k$ and $\bar{y}$ which gives certain guarantees about the estimation of the actual means μ_{x^k} and μ_y (estimated by $\bar{x}^k$ and $\bar{y}$ respectively).

Guaranteeing margin of error m. At the outset, let us make two reasonable assumptions. First, we assume that the samples are normally distributed. This is the common assumption if nothing is known about the actual underlying probability distribution. Second, as suggested in [16], we assume the sample standard deviations which we found in past observations to be the actual standard deviations σ_x^k and σ_y. Then, the following formula gives us the number of challenges n that guarantees a specified margin of error m when estimating μ_{x^k} (or μ_y analogously) [16]:

$$n = \left(\frac{z^* \sigma_x^k}{m} \right)^2 , \tag{23}$$

where the critical value z^* is a constant that determines how confident we can be. Common critical values z^* are 1.645 for 90%, 1.960 for 95% and 2.576 for 99%. More specifically, the integral of the standard normal distribution in the range $[-z^*, z^*]$ equals the respective confidence level. If z^* is for instance chosen for a confidence level of 99%, we know that if we use n challenges for computing $\bar{x}^k$, the actual mean μ_{x^k} will lie in the interval $\bar{x}^k \pm m$ with the probability of 0.99.

Choosing margin of error m. The margin of error m is chosen such that we can be confident that the order of the first two most trustworthy agents is confirmed. In turn, this confirms that the selection of the first agent is the best choice. Let us call the first and the second agent α_1 and α_2 respectively, so we have $T_{\alpha_1} \geq T_{\alpha_2}$. We want to make sure that for the next trust experience this order is not reversed by chance. Recall that a trust experience t_α is defined as the difference between $\bar{y}$ and $\bar{x}^k$ weighted by the sum of the corresponding standard deviations (see formula (22)). As we use $2 * n$ challenges to find $\bar{y}$ and $\bar{x}^k$ respectively, the overall margin of error for the difference of $\bar{y}$ and $\bar{x}^k$ will not be higher than $2 * m$. The largest margin of error m' for which $t_{\alpha_1} \geq t_{\alpha_2}$ is still true (with the given confidence), must therefore fulfill the equation where t_{α_1} takes the lowest and t_{α_2} the highest possible value.

$$t_{\alpha_1} \geq \frac{\bar{y}_1 - \bar{x}_1^k - 2m'}{\underbrace{\sigma_{y_1} + \sigma_{x_1}{}^k}_{=:a}} = \frac{\bar{y}_2 - \bar{x}_2^k + 2m'}{\underbrace{\sigma_{y_2} + \sigma_{x_2}{}^k}_{=:b}} \geq t_{\alpha_2} , \tag{24}$$

where the inner equation can be solved to give:

$$m' = \frac{(t_{\alpha_1} - t_{\alpha_2})ab}{2(a+b)} = \frac{b(\bar{y}_1 - \bar{x}_1^k) - a(\bar{y}_2 - \bar{x}_2^k)}{2(a+b)} . \tag{25}$$

So, a choice of m with the constraint $m \leq m'$, guarantees with the specified confidence that we will get $t_{\alpha_1} \geq t_{\alpha_2}$ — in the case that this is the true order. To limit the number of challenges, we choose the maximal margin of error m that fulfills this constraint, which is given by $m := m'$. We also impose an additional lower bound on m, in order to prevent the number of challenges to grow disproportionally when the differences between the agent's trustworthiness with respect to this specific attack class AC_k are insignificant.

6 Experimental Evaluation

In the experimental part of our work, we evaluate two aspects of the mechanism: its ability to effectively reduce the number of false positives, while relying on an acceptable number of challenges, and its ability to selectively identify the events relevant to the priority threats as specified by the system administrator.

All the experiments were conducted on a university network, on the background of the regular network traffic. This background traffic contains roughly 10% of malicious flows, principally related to scanning, peer-to-peer activity, botnet propagation and brute force attacks on passwords, in no particular order.

In the first series of experiments, we test the ability of the suggested mechanism to produce the classifications with a reasonable error rate as expressed in terms of false positives and false negatives. To evaluate the error rate, we have manually classified the traffic from a significant subset of active hosts on the network. This classified traffic is then used to gauge the effectiveness of the method. The system observed about 80 000 flows every 5 minutes, with roughly 20 000 flows being malicious, and that the evaluation was performed over about seventy 5-minute long observation intervals. The system contained 30 aggregation agents, each of them averaging the opinions of the 5 underlying detection agents as described in Section 2.

In Fig. 8, we can see the number of challenges as it evolves over time. At the beginning, the system works with a fixed number of challenges, in order to let the anomaly detection methods in the detection agents adapt to the traffic. Once all the detection agents start (at step 5, after 25 minutes), the system starts to progressively insert more challenges, in order to build an initial assessment of all classifier agents. The number of challenges peaks at around the step 14, when it reaches 100 (all challenges combined). Once a user agent has built the initial trustworthiness for all agents, the number of challenges decreases until it levels out at around 40 (legitimate and malicious

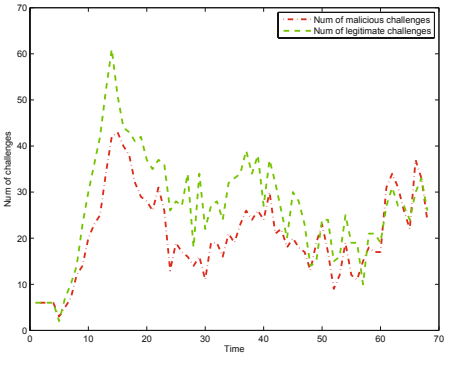


Fig. 8. Number of challenges over time, both legitimate (top, green curve) and malicious (bottom, red curve)

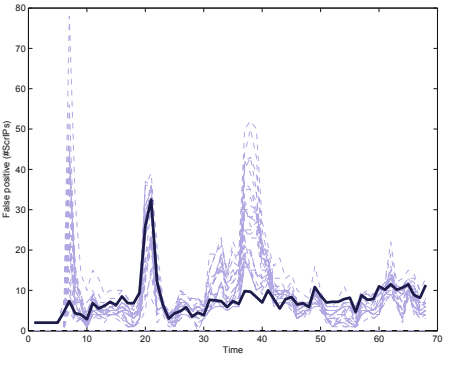


Fig. 9. Number of false positives (unique sources). Each aggregation agent is represented by one thin curve, the solid curve shows the performance of the aggregation agent dynamically selected by the system.

Table 1. Results of static system with arithmetic average (top line) compared to the selection of a single aggregation agent (middle part) and the dynamic self-adaptation mechanism described in this paper. Values are averaged to obtain the expected error numbers for one observation period.

Result	False Negat. [# sources]	False Posit. [# sources]
Arithmetic average	14.7	12.5
Average for aggregation fct.	13.1	24.3
Min FP for aggregation fct.	14.5	5.3
Min FN for aggregation fct.	9.8	125.2
Best aggregation fct.	13.7	5.7
Adaptive aggregation selection	14.0	3.1

challenges combined), where it fluctuates until the end of experiment. However, there are two notable increases to explain: between steps 30 and 40, and after step 60.

These increases can be easily explained when looking at Fig. 9, which shows the number of false positives in terms of unique source IP addresses. During these time intervals, we can notice that the choice of an appropriate aggregation agent has a huge impact on the quality of results, and that the adapted system is able to minimize the number of false positives. The number of challenges is lower between steps 40 and 60, when all agents provide similar results, and increases again around 60, where the performance of the aggregation agents varies somewhat more. On the other hand, we can see that the user agent did not manage to avoid a spike in false positives around the step 20, when it did not yet have a representative trust model.

The results shown in Fig. 9 are summarized in Table 1. We can see that the challenge-based, dynamic adaptation mechanism clearly outperforms the simple arithmetic average aggregation, which is the optimal selection when we have no information regarding the detection agent's performance. It also outperforms any single aggregation function selected using the **a-posteriori** knowledge from the pool of all 30 functions. All the methods have a comparable rate of false negatives, but differ in the rate of false positives, where the dynamic selection clearly outperforms the best aggregation functions. The relatively important margin of separation between the dynamic selection and best false positives of any single aggregation is given by the fact that the dynamic selection can avoid relatively high number of false positives during the periods when the individual aggregation functions differ in performance, such as around the sets 30-40. This further underlines the importance of the adaptive rate of challenge insertion, which allows fast identification of the optimal system output during the changes of system characteristics.

In Fig. 10, we can see the dynamics of the aggregation operator/agent selection over time. With an exception of the initial 6 intervals, when the operator #0 (arithmetic average) is selected by default, the system dynamically selects between the remaining operators, with about half of the selections being the operators #23 and #24. Both these operators include OWA as well as anomaly-detection-method-based weight average portion. They are identical in the fixed part, where they attribute the weight 0.33 to each of the agents Xu [7], MINDS [10] and TAPS [11]. The operators differ in the OWA part, where the first one builds its opinion from the three lowest anomaly values,

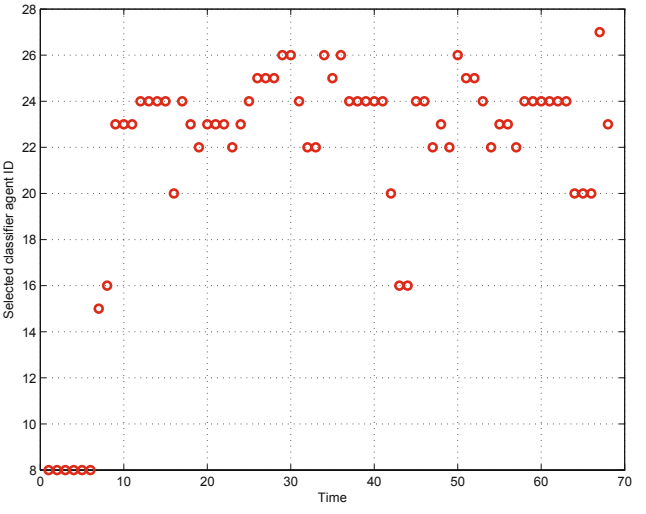


Fig. 10. Selected aggregation agent (identified by the ID number on axis y) for each time step

while the second considers the third and fourth anomaly values. The weights of the detection-method-based averaging and order weighted averaging parts are 0.5 for both operators. It is interesting to note that the system managed to pick the three methods with the most diverse set of anomaly detection features (in the fixed part), consistently with basic ensemble classification [17] principles. The quality of this result is therefore based not only on the absolute quality of underlying detection methods, but also benefits from the diversity of the anomaly detection methods.

In the above-described experiments, the challenges were inserted uniformly, regardless of the attack type. In the following, we will try to measure the effects of challenge insertion in terms of system sensitivity with respect to specific attacks. To do so, we have used the simple server compromise attack tree specified in Fig. 5 to generate the challenges optimizing the system, and we have then attempted to compromise one of the hosts on our network using the standard security tools, such as `nmap` or `metasploit`. The attacks were repeated several times, with changes in speed, tools settings and intensity. We have observed that the system selected the aggregation functions that were able to maximize the likelihood of detection of various stages involved in the server exploit attacks. The anomaly values attributed to horizontal sans, fingerprinting and vertical scans have increased considerably, making them far more likely to be detected. The most dramatic change of behavior was related to the password brute force breaking attempts. These were undetectable with the baseline system configuration, but became detectable with the case-specific system configuration. Buffer overflow attacks were undetectable regardless of the aggregation function, as they are nearly impossible to detect with NBA methods due to the low volumes of traffic involved.

In Table 2, we present the effects of threat model-based adaptation in the traffic used in the first series of experiments. This data set does not match the model at all and provides a good worst case example. We can see that the number of alerts (typically

Table 2. Effects of scenario specific selection on alert numbers in unrelated traffic. Obtained over 72 observation intervals 5 minutes long.

Result [# alerts]	False Negatives	False Positives	True Positives
Neutral challenge insertion	39	201	146
Case-specific insertion	37	249	161

greater than the number of malicious sources used in Table 1) generated by the system has grown, and that the number of false positives increased by about 50. The number of alerts classified as true positives have increased as well (by 15), and the number of false negatives decreased by 2. Note that the total number of alerts is not necessarily identical due to the possible alert fragmentation. Overall, we can see that in order to detect the attacks crucial in the server compromise scenario (e.g. password bruteforcing), the system was able to increase its sensitivity and to find a new equilibrium with different detection profile. It shall be also noted that most of the false positives are repetitive occurrences of traffic structures that are difficult to predict, and that about 80% of them can be eliminated with less than 20 rules in the alert processing engine.

7 Related Work

In literature, more sophisticated formalisms than attack trees have been proposed for modeling attack structures, e.g., attack graphs [18] and attack grammars [19]. However, for our purposes, we do not need to account for the order in which plans of attacks are carried out or the relations between attacks, and hence, the attack tree formalism is sufficiently rich.

In desktop grid computing, spot-checking [20,21] is used to make sure that hosts to which a computation has been outsourced, return correct results. To this end, indistinguishable challenges for which the correct answer is already known are interspersed with actual requests. For a spot-checking approach, where challenges are merged into a vector among a set of real requests, Staab et al. [2] showed how to determine an optimal number of challenges for a given number of real requests. They focused on the case where the answer to a challenge or a real request is binary. This was extended in our work, where we handle the continuous case.

The use of ensemble classification approaches [22] is functionally equivalent to our approach, but with extremely strong assumptions. It requires a pre-classified training data set and don't dynamically adapt system to the changing conditions.

Ghanbari and Amza [23] train belief networks that represent complex systems by injecting failures. At the outset, experts model a belief network that describes the dependencies within a system. The inserted failures then change the prior beliefs of the experts to form better estimates. Through fault injection, the dependencies between the variables in the belief network become evident, and so the overall system can be trained. Opposed to that, we inject challenges to evaluate classification components in terms of accuracy in order to select the most accurate one.

8 Conclusion

Our work presented in this paper aims to close the gap between security policies and formal threat models and the practice of IDS deployment. To achieve this objective, we have designed a runtime adaptation and monitoring framework running on the top of the IDS. It evaluates the performance with respect to the threat models, that are defined as attack trees, with a value assigned to the achievement the objective (root) of the each tree. Objective value can be defined in two manners. In a decision theoretical paradigm, we will aim to minimize our loss by associating an estimate of our loss (or risk) with the achievement of each attack tree root. In the game theoretic model, the value of the attack tree would reflect its value for the attacker. This second option allows us to differentiate between different types of attackers, with different technical capabilities represented by trees with growing complexity and corresponding risk values.

Either type of the threat/risk model can be used as an input for the online monitoring and adaptation process, which is able to evaluate the probability that an attack as defined by the attack tree would pass undetected. This results in an estimate of the **expected undetected loss**, given the current traffic status. This value is also a basis for system adaptation, as the system dynamically reconfigures itself in order to minimize the undetected loss value. The adaptation is based on the evaluation of system response with respect to a set of challenges, pre-classified recorded samples of the past traffic modified to fit the current traffic. The adaptation components of the system use the threat model to define the optimal mix of challenges to insert, in order to align the system performance with the threat models. It is also able to dynamically adjust the number of challenges to insert in response to changing traffic characteristics. The experiments performed with the system show that the dynamic selection of the optimal aggregation function in the CAMNEP system can significantly reduce the number of false positives and that the targeted insertion of challenges selected according to threat models can influence the system sensitivity to reflect the risks associated with each attack type.

The principal limitations of the work are related to the detection capabilities of the individual detection agents aggregated in the system. Using the assumption of classifier diversity [24], we know that the statistical performance of the combined classifier can be significantly better than the performance of individual classifiers. However, the system can not detect (i.e. separate from the traffic) the attacks that none of the individual algorithms can robustly detect.

In our future work, we plan to improve the attack modeling capabilities by inclusion plan-based attack modeling, and to integrate the outputs of the adaptation layer with the alert fusion and correlation capabilities of the system. This combination assess which attack stages are unlikely to be detected, and can use this information to improve the alert correlation [25].

Acknowledgment. This material is based upon work supported by the ITC-A of the US Army under Contract No. W911NF-08-1-0250. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ITC-A of the US Army. Also supported by Czech Ministry of Education grants 6840770038 (CTU) and 6383917201 (CESNET).

References

1. Denning, D.E.: An intrusion-detection model. IEEE Trans. Softw. Eng. 13, 222–232 (1987)
2. Staab, E., Fusenig, V., Engel, T.: Towards trust-based acquisition of unverifiable information. In: Klusch, M., Pěchouček, M., Polleres, A. (eds.) CIA 2008. LNCS (LNAI), vol. 5180, pp. 41–54. Springer, Heidelberg (2008)
3. Rehák, M., Pechoucek, M., Grill, M., Bartos, K.: Trust-based classifier combination for network anomaly detection. In: Klusch, M., Pěchouček, M., Polleres, A. (eds.) CIA 2008. LNCS (LNAI), vol. 5180, pp. 116–130. Springer, Heidelberg (2008)
4. Rehák, M., Pechoucek, M., Bartos, K., Grill, M., Celeda, P., Krmicek, V.: Improving anomaly detection error rate by collective trust modeling. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 398–399. Springer, Heidelberg (2008)
5. Cisco Systems: Cisco IOS NetFlow (2007), `http://www.cisco.com/go/netflow`
6. Scarfone, K., Mell, P.: Guide to intrusion detection and prevention systems (idps). Technical Report 800-94, NIST, US Dept. of Commerce (2007)
7. Xu, K., Zhang, Z.L., Bhattacharrya, S.: Reducing Unwanted Traffic in a Backbone Network. In: USENIX Workshop on Steps to Reduce Unwanted Traffic in the Internet (SRUTI), Boston, MA (2005)
8. Lakhina, A., Crovella, M., Diot, C.: Mining Anomalies using Traffic Feature Distributions. In: ACM SIGCOMM, Philadelphia, PA, pp. 217–228. ACM Press, New York (2005)
9. Lakhina, A., Crovella, M., Diot, C.: Diagnosis Network-Wide Traffic Anomalies. In: ACM SIGCOMM 2004, pp. 219–230. ACM Press, New York (2004)
10. Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.N., Kumar, V., Srivastava, J., Dokas, P.: Minds - minnesota intrusion detection system. In: Next Generation Data Mining. MIT Press, Cambridge (2004)
11. Sridharan, A., Ye, T., Bhattacharyya, S.: Connectionless port scan detection on the backbone, Phoenix, AZ, USA (2006)
12. Yager, R.: On ordered weighted averaging aggregation operators in multicriteria decision making. IEEE Transactions on Systems, Man, and Cybernetics 18, 183–190 (1988)
13. Rubinstein, B.I.P., Nelson, B., Huang, L., Joseph, A.D., Lau, S.-h., Taft, N., Tygar, J.D.: Evading anomaly detection through variance injection attacks on PCA. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 394–395. Springer, Heidelberg (2008)
14. Moore, A.P., Ellison, R.J., Linger, R.C.: Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, CMU Software Engineering Institute (2001)
15. Quine, W.: A way to simplify truth functions. American Mathematical Monthly 62, 627–631 (1955)
16. Moore, D.S.: The Basic Practice of Statistics, 4th edn. W. H. Freeman & Co., New York (2007)
17. Polikar, R.: Esemble based systems in decision making. IEEE Circuits and Systems Mag. 6, 21–45 (2006)
18. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: SP 2002: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Washington, DC, USA, p. 273. IEEE Computer Society, Los Alamitos (2002)
19. Zhang, Y., Fan, X., Wang, Y., Xue, Z.: Attack grammar: A new approach to modeling and analyzing network attack sequences. In: Proc. of the Annual Computer Security Applications Conference (ACSAC 2008), pp. 215–224 (2008)
20. Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. In: CCGRID 2001: Proc. of the 1st Int. Symposium on Cluster Computing and the Grid, Washington, DC, USA, p. 337. IEEE Computer Society, Los Alamitos (2001)

21. Zhao, S., Lo, V., GauthierDickey, C.: Result verification and trust-based scheduling in peer-to-peer grids. In: P2P 2005: Proc. of the 5th IEEE Int. Conf. on Peer-to-Peer Computing, Washington, DC, USA, pp. 31–38. IEEE Computer Society, Los Alamitos (2005)
22. Giacinto, G., Perdisci, R., Rio, M.D., Roli, F.: Intrusion detection in computer networks by a modular ensemble of one-class classifiers. Information Fusion 9, 69–82 (2008)
23. Ghanbari, S., Amza, C.: Semantic-driven model composition for accurate anomaly diagnosis. In: ICAC 2008: Proceedings of the 2008 International Conference on Autonomic Computing, Washington, DC, USA, pp. 35–44. IEEE Computer Society, Los Alamitos (2008)
24. Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000)
25. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2D2: A formal data model for IDS alert correlation. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516, pp. 115–137. Springer, Heidelberg (2002)