On the Effects of Learning Set Corruption in Anomaly-Based Detection of Web Defacements

Eric Medvet and Alberto Bartoli

DEEI, University of Trieste, Via Valerio, Trieste
emedvet@units.it, bartolia@units.it

Abstract. Anomaly detection is a commonly used approach for constructing intrusion detection systems. A key requirement is that the data used for building the resource profile are indeed attack-free, but this issue is often skipped or taken for granted. In this work we consider the problem of corruption in the learning data, with respect to a specific detection system, i.e., a web site integrity checker. We used corrupted learning sets and observed their impact on performance (in terms of false positives and false negatives). This analysis enabled us to gain important insights into this rather unexplored issue. Based on this analysis we also present a procedure for detecting whether a learning set is corrupted. We evaluated the performance of our proposal and obtained very good results up to a corruption rate close to 50%. Our experiments are based on collections of real data and consider three different flavors of anomaly detection.

1 Introduction

Anomaly detection is a powerful and commonly used approach for constructing intrusion detection systems. With this approach the system constructs automatically a profile of the resource to be monitored, starting from a collection of data representing normal usage (the learning set). Once this profile has been established the system signals an anomaly whenever the actual observation of the resource deviates from the profile, on the assumption that any anomalies represent evidence of an attack.

A key requirement is that the learning set is indeed attack-free, otherwise the presence of attacks would be incorporated in the profile and, thus, considered as a normal status. Although this requirement is crucial for the effectiveness of anomaly detection, in practice the absence of attacks in the learning set is either taken for granted or verified "manually". While such a pragmatic approach may be feasible in a carefully controlled environment, it clearly becomes problematic in many scenarios of practical interest. Building a large number of profiles for resources immersed in their production environment, for example, cannot be done by inspecting each learning set "manually" to make sure there were no attacks. A cross-the-fingers approach, on the other hand, can only lead to the construction of potentially unreliable profiles.

In this work we focus on the problem of corruption in the learning set, i.e., a learning set containing records which do not represent a "normal" condition for the observed resource. We think that focussing on this fundamental issue could broaden the scope of anomaly-based detection frameworks. For example, in order to apply anomaly-based monitoring on a large scale—to hundreds or even thousands of resources—it is necessary to build a large number of learning sets, one for each resource to be monitored (e.g., [1] monitored many web applications at the same time). It is clearly not feasible to "manually" check each learning set to make sure there are no attacks hidden in it. The fact that profiles could have to be updated periodically in order to reflect changes in legitimate usage of resources (e.g., [2]) may only exacerbate this problem.

We restrict our analysis to a specific detection system that we developed earlier, i.e., an anomaly-based web site integrity checker [3,4]. This tool is able to monitor many remote web pages automatically using an anomaly detection approach, which may help in triggering prompt reactions in the presence of unauthorized modifications. The tool builds an individual profile for each monitored page, by simply observing that page for a while. The process is fully automatic, i.e., no prior knowledge about content and appearance of the monitored page is required. The scope of our analysis is clearly narrowed by the specific detection system that we consider, but we believe that our approach may be of interest also for other forms of detection systems.

A roadmap to deal with the problem of potentially corrupted learning sets involves: (1) *understanding*, i.e., evaluating quantitatively the effects of a corrupted learning set on the effectiveness of anomaly detection; (2) *detecting*, i.e., being able to discriminate between a corrupted learning set and a clean one; and, possibly, (3) *mitigating*, i.e., preserving an acceptable performance level of the detection system in spite of a corrupted learning set. In this paper we focus on the first two steps. Our experiments are based on collections of real data and consider three different flavors of anomaly detection. We first assess the effects of a corrupted learning set on performance, in terms of false positives and false negatives. This analysis enabled us to gain important insights into this rather unexplored issue. Based on this analysis we also present a procedure for detecting automatically whether a learning set contains corrupted records. This procedure may be used for handling large collections of datasets automatically, raising an alert to a human operator only for those which look suspicious (much like the alerts raised after the learning phase has completed). We evaluated the performance of this procedure and obtained very good results up to a corruption rate close to 50%.

2 Related Work

Broadly speaking, anomaly detection is an instance of *inductive learning classification* in which the goal is to build a profile able of discriminating between only two classes—i.e., normal and anomalous—using learning data corresponding to a single class—i.e., normal [5,6,7,8]. In the inductive machine learning field the corruption of learning set is indicated as *noise*, which is generally subdivided in two categories: attribute noise and class noise. The former consists in records for which one or more attributes are not really representative of the corresponding class, whereas the latter concerns records of the dataset which are wrongly labeled. A comparison between the effects of the two different types of noise on classifier accuracy is presented in [9]; our work refers to a very specific instance of the inductive machine learning problem and considers only class noise.

Concerning class noise, there is a substantial amount of work proposing solutions for identifying and then removing the corrupting (mislabled) records. More in general, this issue can be addressed with *outlier detection* techniques [10]. We have not investigated whether such techniques can be applied to our framework, that is characterized by a small learning set—usually a few tens of records. In this work we are merely concerned with the problem of detecting whether the learning set is indeed clean or contains some amount of class noise.

An important method for finding mislabeled records in the learning set is given in [11]. The idea consists in building a set of filtering classifiers from only part of the learning set and then testing whether data in the remaining part also belong to the profile. The learning set is partitioned in a number of subsets and, for each subset, a filtering classifier is trained using the remaining part of the learning set. Each record of the learning set is then input to each of the filtering classifiers. The cited paper proposes and evaluates several criteria for merging the labels generated by the filtering classifiers. The method should be able to identify outliers regardless of the specific classifier being used, hence, regardless of the chosen model for the data. A very similar approach, specifically tailored to large datasets, is proposed in [9]. Our work differs from these proposals in that we do not partition the learning set in smaller sets. This can be an advantage in the cases—our test scenario is indeed one of them—where learning sets may be very small and thus a further division will lead to a ineffective classifier: such cases are considered by Forman and Cohen [12] in their comparative study about machine learning applied to small learning set.

Concerning specifically anomaly detection, we are not aware of works covering both the understanding and detecting phases of the problem of corrupted learning sets. We are only aware of a few published experiments about the effects of a corrupted learning set—i.e., only the understanding phase. Hu et al. [13] consider an anomaly-based host intrusion detection system and compare the performance of Robust Support Vector Machines (RSVMs), conventional Support Vector Machines and nearest neighbor classifiers using 1998 DARPA BSM data set. Besides experimenting with clean learning sets, they consider also artificially corrupted learning datasets and find that RSVM are more robust to noise.

A similar analysis is proposed by Mahoney and Chan [14]. The authors present an IDS which detects anomalies in packet header fields. In addition to the normal effectiveness evaluation (with 1999 DARPA dataset), they experiment also with smaller and corrupted learning sets. They motivate this choice based on the practical difficulty in obtaining attack-free learning sets. The authors test their tool in a real environment and retune the tool every day based on the data collected on the previous day. A significant loss in detection rate is highlighted, but the relation between loss and corruption is not analyzed because the corruption level is not measured accurately.

A radically different approach to the problem of corrupted learning sets is that of *unsupervised anomaly detection*. With these techniques learning set records do not need to be labelled as clean or attack-related and the detection method itself is intrinsically robust to a certain amount of noise. Along this line, Laskov et al. [15] present a network IDS based on a formulation of a one-class Support Vector Machine (SVM) [16], whereas Wang and Stolfo [17] present a network IDS where anomalies with respect to the profile are based on the the Mahalanobis distance. These techniques usually work on learning datasets much larger than ours, which often consists of only a few tens of records.

3 The Test Scenario: Web Site Defacement Detection

3.1 Motivation and Framework

Our analysis is based on a specific detection system that we developed earlier, i.e., an anomaly-based web site integrity checker [3]. This tool is aimed at monitoring the integrity of remote web pages automatically while remaining fully decoupled from them, in particular, without requiring any prior knowledge about content or appearance of the monitored resources. The tool builds an individual profile for each monitored page, by simply observing that page for a while, and signals an anomaly whenever a page deviates from its profile. Full details can be found in [4], in particular concerning performance, limitations and open issues.

The issue of learning set corruption is particularly significant in this scenario for two key reasons. First, the learning set for each page is usually small. We decided that the monitoring of a new page should require the gathering of learning data for a few days at most. Since readings taken every few minutes usually exhibit very little difference, the result is that a learning set typically consists of a few tens of readings. It follows that the learning set is so small that even a little amount of corruption (e.g., a single corrupting reading) could have significant impact. Second, it is necessary to update the profile periodically [4]. Assuming a visual inspection of each new learning set is clearly not an option: the tool must be able to retune itself automatically.

Although we focus on a web site integrity checker, however, we considered a more general anomaly-based framework. We consider a source of information producing a sequence of *readings* $\{i_1, i_2, \dots\}$ which is input to a *detector* (Fig. 1). The detector will classify each reading as being either *normal* or *anomalous*. The detector consists internally of a *refiner* followed by an *aggregator*, both described in the next sections. In our scenario the source of information is a web page, univocally identified by an URL, and each reading consists of the document downloaded from that URL.

Detector Architecture. The refiner implements a function that takes a reading i and produces a fixed size numeric vector $v = R(i)$. The details of the

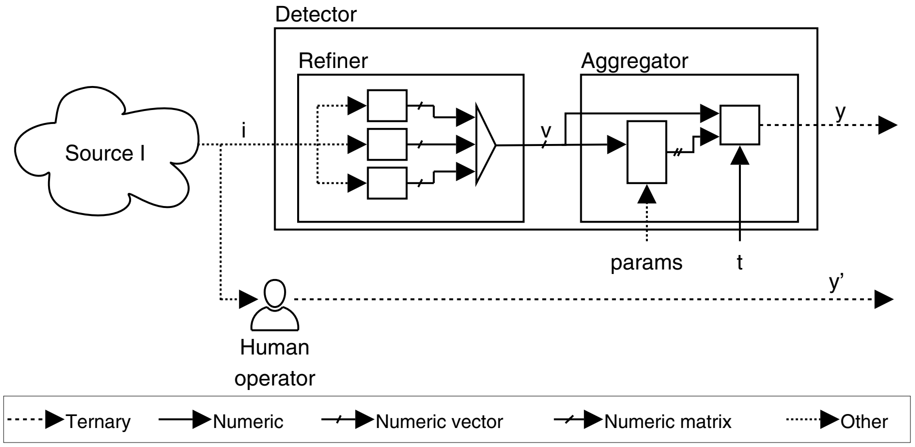


Fig. 1. Detector architecture. Different arrow types correspond to different types of data.

transformation are problem specific. In our case the transformation involves evaluating and quantifying many features of a web page related to both its content and appearance. The refiner is internally composed by one or more *sensors.* A sensor S is a component which receives as input the reading i and outputs a fixed size vector of real numbers v_S. The output of the refiner is composed by concatenating the output of all sensors. Sensors are functional blocks and have no internal state: $v = R(i)$ depends only on the current input i and does not depend on any prior reading. In our case we consider a refiner producing a vector $v = R(i)$ of 1466 elements, obtained by concatening the outputs from 43 different sensors (Sect. 3.2).

The aggregator is the core component of the detector and it is the one that actually implements the anomaly detection. In a first phase, that we call the *learning phase*, the aggregator collects a number of readings in order to build the profile of the resource; during this phase, the aggregator is not able to classify readings. In a second phase, the *monitoring phase*, the aggregator compares the current reading against the profile and considers it as anomalous whenever it is too much different from the profile. The output in the monitoring phase depends on an external parameter called *normalized discrimination threshold* (threshold for short), which affects the sensitivity-specificity tradeoff of the detector. We denote the threshold by t and the output of the refiner for reading i_k by v_k.

The aggregator performs the *learning phase* on a learning sequence obtained with the first l readings $\{v_1, \dots, v_l\}$, that we denote S_{learning}. With this procedure the aggregator sets the values for some internal numeric parameters $\{p_1, \dots, p_m\}$, that constitute the profile P of the resource. During the learning phase, the output y_k for each reading i_k is always $y_k = \text{unable}$, meaning that the aggregator is currently unable to classify the reading.

After the learning phase, the aggregator enters the *monitoring phase*, in which it considers the monitoring sequence obtained with the remaining readings $\{v_{l+1}, v_{l+2}, \dots\}$, that we denote $S_{\text{monitoring}}$. In this phase the aggregator compares each reading against the profile P established in the learning phase. The output y_k for each reading i_k is given by a function $F^{A}_{\text{compare}}(v_k, P, t)$ that may return either y_k = negative (meaning the reading is normal) or y_k = positive (meaning the reading is anomalous).

We built 3 different aggregators, which differ in the way they exploit application-specific knowledge in order to elaborate the outputs produced by the refiner. The details are presented in the next sections.

3.2 Prototype Details

Sensors. The 43 sensors contained in our refiner can be grouped in 5 categories, based on the way they extract information from readings. A brief description of each category follows. Table 1 summarizes salient information about sensor categories and indicates the number of sensors and the corresponding size for the vector v portion in each category.

Table 1. Sensor categories and corresponding vector portion sizes

Category	Number of sensors	Vector size
Cardinality	25	25
RelativeFrequencies	2	117
HashedItemCounter	10	920
HashedTree	2	200
Signature	4	4
Total	43	1466

Cardinality sensors. Each sensor in this category outputs a vector composed by only 1 element v^1. The value of v^1 corresponds to the measure of some simple feature of the reading (e.g., the number of lines).

The features taken into account by the sensors of this category are:

- Tags: block type (e.g., the output v^1 of the sensor is a count of the number of block type tags in the reading), content type, text decoration type, title type, form type, structural type, table type, distinct types, all tags, with `class` attribute;
- Size attributes: byte size, mean size of text blocks, number of lines, text length;
- Text style attributes: number of text case shifts, number of letter-to-digit and digit-to-letter shifts, uppercase-to-lowercase ratio;
- Other items: images (all, those whose names contain a digit), forms, tables, links (all, containing a digit, external, absolute).

RelativeFrequencies sensors. Each sensor S in this category outputs a vector composed by n_S elements $v = |v^1, \dots, v^{n_S}|$. Given a reading i, S computes the relative frequency of each item in the item class analyzed by S (e.g., lowercase letters), whose size is known and equal to n_S. The value of the element v^k is equal to the relative frequency of the k -th item of the given class.

This category includes two sensors. One analyzes lowercase letters contained in the visible textual part of the resource ($n_S = 26$); the other analyzes HTML elements of the resource—e.g., `HTML`, `BODY`, `HEAD`, and so on—with $n_S = 91$.

HashedItemsCounter sensors. Each sensor S in this category outputs a vector composed by n_S elements $v = |v^1, \dots, v^{n_S}|$ and works as follows. Given a reading i, S: (1) sets to 0 each element v^k of v; (2) builds a set $L = \{l_1, l_2, \dots\}$ of items belonging to the considered class (e.g., absolute linked URLs) and found in i; note that L contains no duplicate items; (3) for each item l_j, applies a hash function to l_j obtaining a value $1 \leq k_j \leq n_S$; (4) increments v^{k_j} by 1.

This category includes 10 sensors, each associated with one of the following item classes: image URLs (all images, only those whose name contains on or more digits), embedded scripts, tags, words contained in the visible textual part of the resource and linked URLs. The link feature is considered as 5 different sub-features, i.e., by 5 different sensors of this group: all external, all absolute, all without digits, external without digits, absolute without digits. All of the above sensors use a hash function such that $n_S = 100$, except from the sensor considering embedded scripts for which $n_S = 20$. Note that different items could be hashed on the same vector element. We use a large vector size to minimize this possibility, which cannot be avoided completely however.

HashedTree sensors. Each sensor S in this category outputs a vector composed by n_S elements $v = |v^1, \dots, v^{n_S}|$ and works as follows. Given a reading i, S: (1) sets to 0 each element v^k of v; (2) builds a tree H by applying a sensor-specific transformation on the HTML/XML tree of i (see below); (3) for each node $h_{l,j}$ of the level l of H, applies a hash function to $h_{l,j}$ obtaining a value $k_{l,j}$; (4) increments $v^{k_{l,j}}$ by 1. The hash function is such that different levels of the tree are mapped to different adjacent partitions of the output vector v, i.e., each partition is "reserved" for storing information about a single tree level.

This category includes two sensors, one for each of the following transformations:

- Each start tag node of the HTML/XML tree of reading i corresponds to a node in the transformed tree H. Nodes of H contain only the type of the tag (for example, `TABLE` could be a node of H, whereas `<TABLE CLASS="NAME">` could not).
- Only nodes of the HTML/XML tree of reading i that are tags in a predefined set (`HTML`, `BODY`, `HEAD`, `DIV`, `TABLE`, `TR`, `TD`, `FORM`, `FRAME`, `INPUT`, `TEXTAREA`, `STYLE`, `SCRIPT`) correspond to a node in the transformed tree H. Nodes of H contain the full start tag (for example, `<TD CLASS="NAME">` could be a node of H, whereas `<P ID="NEWS">` could not).

Both sensor have $n_S = 200$ and use $2, 4, 50, 90$ and 54 vector elements for storing information about respectively tree levels $1, 2, 3, 4$ and 5; thereby, nodes of level 6 and higher are not considered.

Signature sensors. Each sensor of this category outputs a vector composed by only 1 element v^1, whose value depends on the presence of a given attribute. For a given reading i, $v^1 = 1$ when the attribute is found and $v^1 = 0$ otherwise.

This category includes 4 sensors, one for each of the following attributes (rather common in defaced web pages):

– has a black background;
– contains only one image or no images at all;
– does not contain any tags;
– does not contain any visible text.

Aggregators. As observed in Sect. 3.1, we built 3 different aggregators. They exploit different levels of application-specific knowledge and are described in the next sections. Recall that the learning sequence consists of a sequence $S_{\text{learning}} = \{v_1, \ldots, v_l\}$ obtained from the first l readings, where each v_k is a vector with 1466 elements.

TooManyFiringElements. This aggregator does not exploit any application-specific knowledge. A reading is labelled as anomalous whenever too many elements of v_k are too much different from what expected.

During the learning procedure the aggregator computes the mean η_i and standard deviation σ_i for each element v_k^i, across all vectors in $S_{\text{learning}} = \{v_1, \ldots, v_l\}$. During the monitoring phase the aggregator counts the number of *firing elements*, i.e., those elements whose value is too much different from what expected. An element fires when its value v^i is such that $|v^i - \eta_i| \geq 3\sigma_i$. If the number of firing elements is at least Nt, the reading is classified as anomalous ($N = 1466$ is the size of each vector, t is the threshold).

Note that this aggregator handles all vector elements in the same way, irrespective of how they have been generated by the refiner. Thus, for example, elements generated by a signature sensor are handled in the same way as those generated by hashed. Moreover, the aggregator does not consider any information possibly associated with pairs or sets of elements, i.e., elements generated by either the same sensor or by sensors in the same category.

TooManyFiringSensors. This aggregator exploits some degree of domain-specific knowledge: it "knows" that the vector elements are partitioned in slices and each slice corresponds to a specific sensor. The profile constructed in the learning phase is also partitioned, with one partition associated with each sensor.

In the monitoring phase this aggregator transforms each slice in a boolean, by applying a sensor-specific transformation that depends on the profile (i.e., on the partition of the profile associated with that sensor). When the boolean obtained from a slice is true, we say that the corresponding sensor *fires.* If the number of sensors that fire is at least Mt, the reading is classified as anomalous ($M = 43$ is the number of sensors, t is the threshold).

We describe the details of the learning phase and monitoring phase below. All sensors in the same category are handled in the same way.

Cardinality. In the learning procedure the aggregator determines mean η and standard deviation σ of the values $\{v_1^1, \dots, v_l^1\}$—recall that Cardinality sensors output a vector composed by a single value. In the monitoring phase a sensor fires if its output value v^1 is such that $|v^1 - \eta| \geq 3\sigma$.

RelativeFrequencies. A sensor in this category fires when the relative frequencies (of the class items associated with the sensor) observed in the current reading are too much different from what expected. In detail, let n_S be the size of the slice output by a sensor S. In the learning phase, the aggregator performs the following steps: (i) evaluates the mean values $\{\eta_1, \dots, \eta_{n_S}\}$ of the vector elements associated with S; (ii) computes the following for each reading v_k of the learning sequence ($k \in [1, l]$):

$$d_k = \sum_{i=1}^{n_S} |v_k^i - \eta_i| \tag{1}$$

(iii) computes mean η and standard deviation σ of $\{d_1, \dots, d_l\}$.
In the monitoring phase, for a given reading v, the aggregator computes:

$$d = \sum_{i=1}^{n_S} |v^i - \eta_i| \tag{2}$$

The corresponding sensor fires if and only if $|d - \eta| \geq 3\sigma$.

HashedItemsCounter. Let n_S be the size of the slice output by a sensor S. In the learning procedure, the aggregator computes for each slice element the minimum value across all readings in the learning sequence, i.e. $\{m_1, \dots, m_{n_S}\}$. In the monitoring phase S fires if and only if at least one element v^i in the current reading is such that $v^i < m_i$.
The interpretation of this category is as follows. Recall that each slice element is a count of the number of times an item appears in the reading (different items are hashed to different slice elements). Any non-zero element in $\{m_1, \dots, m_{n_S}\}$, thus, corresponds to items which appear in every reading of the learning sequence. In the monitoring phase the sensor fires when there is at least one of these “recurrent items” missing from the current reading.

HashedTree. Sensors in this category are handled in the same way as those of the previous category, but the interpretation of a firing is slightly different. Any non-zero element in $\{m_1, \dots, m_{n_S}\}$ corresponds to a node which appear in every reading of the learning sequence, at the same level of the tree. In the monitoring phase the sensor fires when a portion of this “recurrent tree” is missing from the current reading (i.e., the sensor fires when the tree corresponding to the current reading is not a supertree of the recurrent tree). We omit further details for simplicity, as they can be figured out easily.

Signature. A sensor in this category fires when its output is 1. Recall that these sensors output a single element vector, whose value is 1 whenever they find a specific attribute in the current reading.

As an aside, note that not only the aggregator exploits domain-specific knowledge, it also exploits knowledge about the refiner (e.g., regarding the number of sensors and size of each slice).

TooManyFiringGroups. This aggregator works similarly to the previous one. It transforms slices into boolean values in the same way as above. However, rather then considering all sensors as being equivalent, this aggregator "knows" that sensors are grouped in categories. If the number of categories with at least one sensor that fires is at least Kt, the reading is classified as anomalous ($K = 5$ is the number of categories, t is the threshold).

As we noticed in our previous works, sensors belonging to the same category tend to exhibit a similar behaviour in terms of false positives [3,4]. This aggregator thus exploits domain-specific knowledge more deeply than the previous one.

4 Experiments

4.1 Dataset

In order to perform our experiments, we built a dataset as follows. We observed 15 web pages for about one year, collecting a reading for each page every 6 hours, thus totalling about 1350 readings for almost each web page. These readings compose the *negatives sequences*—one negative sequence S_N for each page: we visually inspected them in order to confirm the assumption that they are all genuine, that is, none of them was a defacement. Table 2 presents a list of the observed pages, which includes pages of e-commerce web sites, newspapers web sites, and alike. Pages differ in size, content and dynamicity and are the same that we observed for a shorter period in [4].

Then we built a single *positive sequence* S_P composed by 100 readings extracted from a publicly available defacement archive.[1] Defacements composing S_P were not related with any of the 15 resources that we observed—as pointed out above none of these resources was defaced during our monitoring period. The next section explains how we used S_P readings in order to simulate attacks to the monitored resources.

4.2 Methodology

We wanted to gain insight about how a given aggregator A copes with a corrupted learning sequence S_{learning}, i.e., when S_{learning} contains readings that must be classified as anomalous. We considered different *corruption rates* r, i.e.,

[1] Our selection is available online at `http://www.units.it/bartolia/download/BartoliMedvetAttackSet.zip`

Table 2. List of web resources composing our dataset. Change frequency is a rough approximation of how often non minor changes were applied to the resource, according to our observations. Concerning Amazon – Home page and Wikipedia – Random page, we noted that most of the content section of the resource changed at every reading, independently from the time.

	Change frequency	Monitoring period	# of readings
Amazon – Home page	Almost every reading	9/19/05–9/1/06	1340
Ansa – Home page	Every 4–6 hours	9/19/05–9/1/06	1340
Ansa – Rss sport	Every 4–6 hours	9/19/05–9/1/06	1340
ASF France – Home page	Weekly	9/19/05–9/1/06	1340
ASF France – Traffic page	Less than weekly	9/19/05–9/1/06	1340
Cnn – Businnes	Every 4–6 hours	9/19/05–9/1/06	1340
Cnn – Home page	Every 4–6 hours	9/19/05–9/1/06	1340
Cnn – Weather	Daily	9/19/05–9/1/06	1340
Java – Top 25 bugs	Less than weekly	12/1/05–9/1/06	1096
Repubblica – Home page	Every 4–6 hours	9/19/05–9/1/06	1340
Repubblica – Tech. and science	Every 2–3 days	9/19/05–9/1/06	1340
The Server Side – Home page	Every 2–3 days	12/1/05–9/1/06	1095
The Server Side – Tech talks	Weekly	12/1/05–9/1/06	1096
Univ. of Trieste – Home page	Weekly	9/19/05–9/1/06	1342
Wikipedia – Random page	Every reading	9/19/05–9/1/06	1337

different fractions of positive reading in S_{learning}. We measured A effectiveness, in terms of false positive rate (FPR), false negative rate (FNR) and area under the ROC curve (A_{ROC}).

For each aggregator A, corruption rate r and page p:

- We constructed a learning sequence S_{learning} as follows. (1) We extracted a sequence $S = \{i_0^n, \dots, i_{125}^n\}$ of 125 consecutive readings from the negative sequence S_N of the page p. (2) We split S in two subsequences: S'_{learning} composed by the first $l = 50$ readings and S'_{test} composed by the last 75 readings. (3) We constructed S_{learning} by replacing the $50 \cdot r$ final readings of S'_{learning} with a positive reading extracted from the positive sequence S_P and repeated $50 \cdot r$ times (for simulating a defacement occurred while collecting the learning data). Note that r represents the percentage of S_{learning} that is corrupted.
- We constructed a test sequence S_{test} for evaluating the profile built with the corrupted learning sequence S_{learning} as follows. (4) We inserted at the beginning the sequence S'_{test} obtained at the previous step. (5) We appended 75 different positives extracted at random from the positive sequence S_P (and including the one that corrupted S_{learning}). In other words S_{test} is composed of 150 readings, the expected output should be negative in the first half and positive in the second half.

We repeated the above experiment several times, with $N_S = 10$ different negative sequences S at step 1 and with $N_p = 10$ different positive readings at step 3.

For each tuple $\langle \text{page } p, \text{aggregator } A, \text{corruption rate } r\rangle$, thus, we performed $N_S N_p = 100$ experiments evaluating FPR, FNR and A_{ROC} in each experiment. The values in the next sections are the average values obtained across all 15 pages in our dataset.

5 Results

5.1 Uncorrupted Learning Sequence

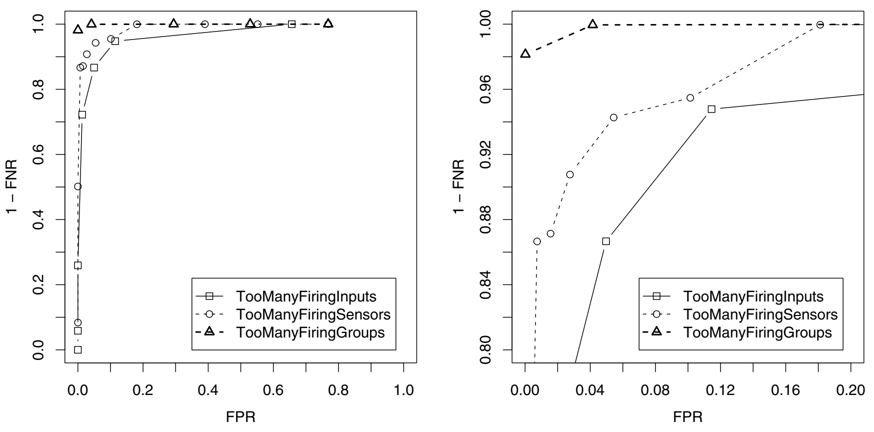
In this section, we present the results obtained for the 3 aggregators presented in Sect. 3.1 evaluated on an uncorrupted learning sequence, i.e., with $r = 0$. The values obtained for FPR, FNR and A_{ROC} in these conditions will serve as a comparison baseline.

Table 3 shows FPR and FNR for the 3 aggregators as a function of the threshold t. The same results are plotted in Fig. 2 in the form of ROC curves. It is clear that TooManyFiringGroups outperforms the other aggregators. In particular, Tab. 3 confirms that, with $t = 0.9$, TooManyFiringGroups never misclassifies a negative reading as positive, while it wrongly undetects only 1.9% of positive readings (i.e., defacements). Such values can not be obtained with any t value for the two other aggregators. This results confirms the intuition that domain-specific knowledge may be very beneficial in the design of an aggregator (this is similar to, e.g., choosing values for conditional probabilities in a Bayesian network [18]).

Table 3. FPR and FNR for the 3 aggregators and several t values obtained with $r = 0$, i.e., with uncorrupted learning sequences, expressed in percentage. Values corresponding to $t = t_{\text{opt}}$ for each aggregator are bolded (see below).

Aggregator (A)	t	0.01	0.05	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.90	0.95
TooManyFiringGroups	FPR	-	-	76.8	-	52.9	-	29.4	-	4.2	**0.0**	-
	FNR	-	-	0.0	-	0.0	-	0.0	-	0.0	**1.9**	-
TooManyFiringSensors	FPR	76.8	55.2	39.0	18.1	10.1	**5.4**	2.8	1.6	0.7	0.0	0.0
	FNR	0.0	0.0	0.0	0.0	4.5	**5.7**	9.2	12.9	13.3	49.8	91.6
TooManyFiringInputs	FPR	65.7	**11.4**	5.0	1.3	0.0	0.0	0.0	-	-	-	-
	FNR	0.0	**5.2**	13.3	27.8	74.1	94.1	100.0	-	-	-	-

The choice of the threshold t depends on the desired trade-off between FPR and FNR. We have emphasized in bold in Tab. 3 the values corresponding to the lowest value of FPR + FNR—just one of the possible performance indexes. We observe that the 3 aggregators have different values for the respective optimal threshold t_{opt}. Assigning the same weight to false positive and false negatives may or may not be appropriate in all scenarios. For example, in the web site defacement detection scenario, one could tolerate some false positive in order to be sure of not missing any defacement. On the contrary, in the spam detection problem, one could accept some undetected spam message while not tolerating a genuine e-mail being thrown away.



(a) ROC curves (b) ROC curves (detail)

Fig. 2. ROC curvers for the 3 aggregators obtained with $r = 0$, i.e., with uncorrupted learning sequences. The plot on the right shows the area with FPR and FNR lower than 20%.

5.2 Corrupted Learning Sequence

In this section, we present the results concerning the effectiveness of the aggregators when corrupted learning sequences are used. We experimented with the following values for the corruption rate r, including 0 (the uncorrupted sequence): $0, 0.02, 0.05, 0.1, 0.2, 0.35, 0.5, 0.75$. Being $l = 50$ the size of the learning sequence, these rates mean that respectively $0, 1, 3, 5, 10, 18, 25, 38$ positive readings have been inserted in the learning sequence.

Table 4 shows FPR and FNR for the 3 aggregators with varying values of the corruption rate r. These results refer to the optimal threshold t_{opt} determined as explained above for $r = 0$.

Not surprisingly, increasing the corruption rate results in an increment of FNR for each aggregator. In other words, the more corrupted the learning sequence, the less sensitive to attacks the aggregator. Increasing the corruption rate also results in a decrease of FPR, due to the fact that the learning sequence, and hence the profile, becomes less and less page-specific. Although performance appears to quickly become unacceptable, varying the threshold may greatly help, as clarified in the following. The reason is because the above data corresponds to a threshold t that is optimal for an *uncorrupted* learning sequence, but this value is not necessarily optimal for a *corrupted* one.

A more general characterization of the performance of each aggregator is given in Fig. 3, which plots A_{ROC} as a function of the corruption rate r. As such, each point in this graph provides a performance index capturing *all* possible values for the threshold t. We found that A_{ROC} does *not* decrease monotonically when the corruption rate increases. On the contrary there is an A_{ROC} increase

Table 4. FPR and FNR, obtained with $t = t_{\text{opt}}$ (see Tab. 3) and presented in percentage, for the aggregator with different values for r

	TooManyFiringGroups		TooManyFiringSensors		TooManyFiringInputs	
	FPR	FNR	FPR	FNR	FPR	FNR
r	$t = t_{\text{opt}} = 0.9$		$t = t_{\text{opt}} = 0.4$		$t = t_{\text{opt}} = 0.05$	
0.00	0.0	1.9	5.4	5.7	11.4	5.2
0.02	0.0	73.4	0.1	12.4	6.5	5.6
0.05	0.0	77.6	0.1	12.9	5.5	6.3
0.10	0.0	83.9	0.0	69.5	2.5	75.8
0.20	0.0	87.5	0.1	99.9	2.5	99.4
0.35	0.0	86.7	0.1	99.9	2.6	99.6
0.50	0.0	77.6	0.1	99.7	2.8	99.6
0.75	0.0	87.7	0.3	99.8	11.3	99.4

when $r < 0.05$, the entity of the improvement being dependent on the specific aggregator. In other words, a very small corruption in the learning sequence is beneficial for all aggregators from the A_{ROC} point of view. This suggests that under a modest corruption there is some point of the ROC curve of each aggregator—i.e., some value of t—for which FPR and FNR are acceptable and, maybe, even slightly better than with an uncorrupted learning sequence. Of course, finding that point would require the knowledge of the corruption rate. The slight increase in A_{ROC} is probably due to the fact that a small amount of noise (i.e., corruption) may balance the overfitting effect, which affects negatively FPR and may be an issue in pages that are less dynamic and whose corresponding learning sets are hence less representative.

The relation between FPR + FNR and t is shown in Fig. 4, which plots one curve for each corruption rate. The lowest point of each curve corresponds to $t = t_{\text{opt}}$. We remark again that one could choose to minimize a different function of FPR and FNR. The essential issues of our arguments would not change, however. Figure 4 illustrates two important facts. First, the optimal threshold depends on the corruption rate. That is, a threshold optimized for an uncorrupted learning sequence is not necessarily the best threshold for a corrupted learning sequence. Second, performance with a corrupted learning sequence are not necessarily worse than with an uncorrupted learning sequence. For example, decreasing t improves the FPR + FNR index significantly, especially for the TooManyFiringSensors aggregator. Table 5 summarizes the improvements exhibited by the aggregators using the optimal value t_{opt} for two salient r values.

A key lesson from these experiments is that the aggregators may remain practically useful (i.e., they exhibit acceptable FPR and FNR) even in the presence of a moderate degree of corruption in the learning sequence. The problem is, turning this observation into a practically usable procedure is far from being immediate: one should know the corruption rate in order to select a suitable working point, but this is precisely the unknown entity.

Table 5. Aggregators optimal working point for two different corruption rates and corresponding FPR and FNR, presented in percentage

Aggregator (A)	r	t_{opt}	FPR	FNR	FPR + FNR
TooManyFiringGroups	0.0	0.9	0.0	1.9	1.9
	0.05	0.5	0.2	0.8	1.0
TooManyFiringSensors	0.0	0.4	5.4	5.7	11.1
	0.05	0.1	1.5	0.0	1.5
TooManyFiringInputs	0.0	0.05	11.4	5.2	16.6
	0.05	0.05	5.5	6.3	11.8

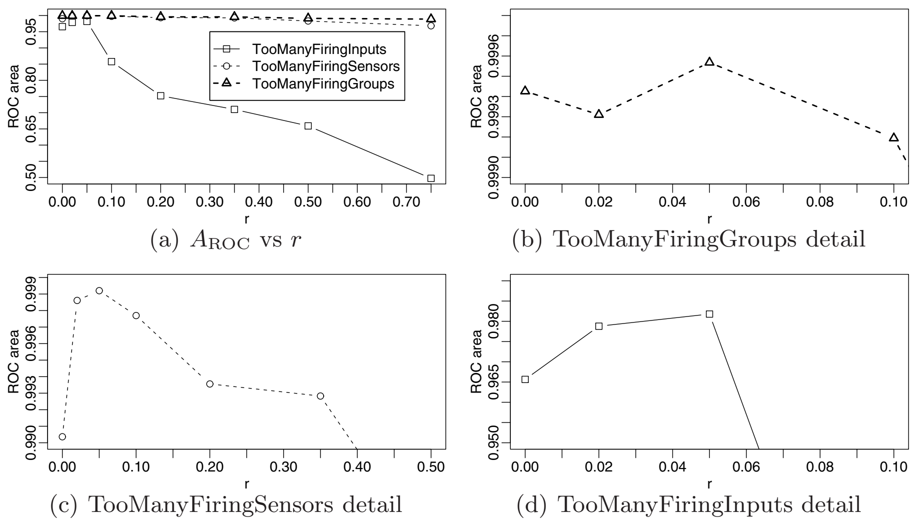
6 A Corruption Detection Procedure

We have seen in the previous section that the impact of the corruption rate r on FPR and FNR is not linear. In particular, it can be observed that the change in FPR and FNR is much sharper when r increases from 0 to 0.02 than when r increases from 0.02 to 0.05 (see Tab. 4). The effects on performance, thus, are much stronger when switching from a clean learning sequence to a corrupted learning sequence than with a moderate increase of a (non-zero) corruption rate. We performed a number of experiments, not shown here for space reasons, to verify that this phenomenon does occur in a broad range of operating conditions. We exploited the above observation for building a simple yet effective corruption detection procedure, which is presented below.

6.1 Description

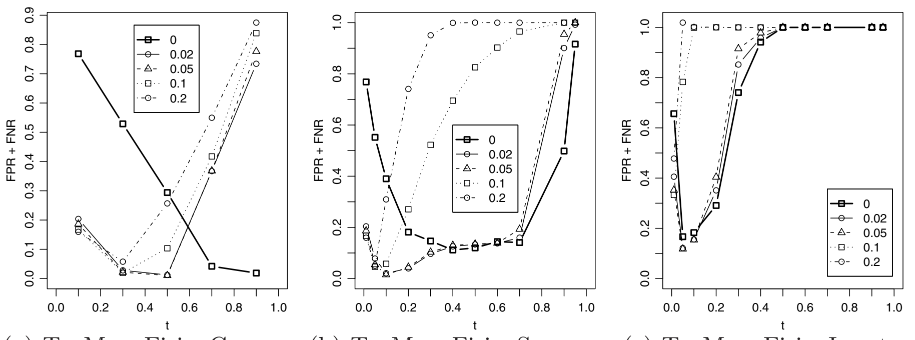
The objective is to determine whether a given learning sequence S^0_{learning} is corrupted. The key idea is quite simple. We build three profiles, one with S^0_{learning} and the other with two learning sequences obtained by artificially corrupting S^0_{learning} with 1 or 3 positive readings. Then we measure performance of the three profiles on a *same* sequence S_{check}. If we observe a strong change in FPR and/or FNR when switching from the first profile to the other two profiles, then S^0_{learning} was probably clean, otherwise it was probably already corrupted.

In detail, let $S'_P = \{i^p_0, \dots, i^p_n\}$ be a set of n positive readings. We construct S_{check} with a mixture of genuine readings and positive readings. Then we proceed as follows. (1) We tune the aggregator on S^0_{learning}; we measure FPR^0 and FNR^0 on the check sequence S_{check}; (2) For a given i^p_i in S'_P, we construct two learning sequence $S^{1,i}_{\text{learning}}$ and $S^{3,i}_{\text{learning}}$ by replacing respectively 1 and 3 random readings of S^0_{learning} with i^p_i; we tune the aggregator on these learning sequences; we measure the corresponding performance on S_{check} ($\text{FPR}^{1,i}$, $\text{FNR}^{1,i}$, $\text{FPR}^{3,i}$ and $\text{FNR}^{3,i}$). (3) We repeat the previous step for each i^p_i of S'_P and evaluate mean and standard deviation of the performance indexes.



(a) A_{ROC} vs r (b) TooManyFiringGroups detail (c) TooManyFiringSensors detail (d) TooManyFiringInputs detail

Fig. 3. The area under the ROC curve (A_{ROC}) vs the corruption rate r. Figures 3(b), 3(c) and 3(d) show salient A_{ROC} values for the 3 aggregators separately.



(a) TooManyFiringGroups (b) TooManyFiringSensors (c) TooManyFiringInputs

Fig. 4. Effectiveness of aggregators, as sum of FPR and FNR, plotted vs the aggregator normalized discrimination threshold t, for different modest corruption rates including $r = 0$. The lowest point of each curve corresponds to the optimal working point $t = t_{\mathrm{opt}}$ in the given conditions.

The original learning sequence S^0_{learning} is deemed corrupted if and only if at least one of the following holds:

$$\mathrm{FPR}^0 - \mathrm{FPR}^\eta \geq m\mathrm{FPR}^\sigma \tag{3}$$

$$\mathrm{FNR}^0 - \mathrm{FNR}^\eta \geq m\mathrm{FNR}^\sigma \tag{4}$$

where m corresponds to a sensitivity parameter of the procedure.

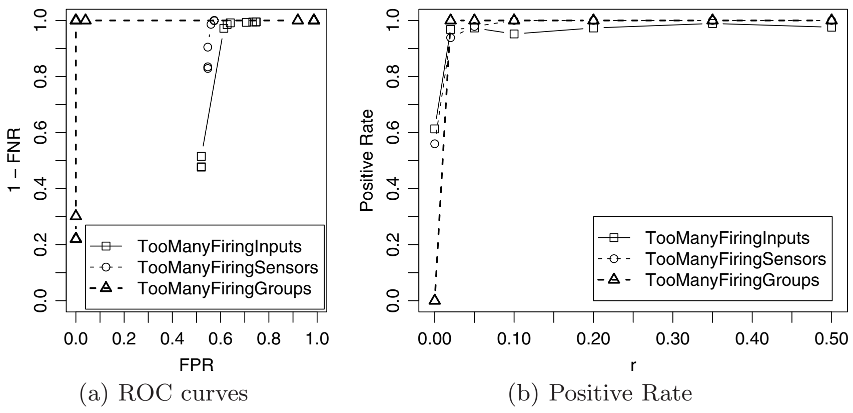
6.2 Evaluation and Results

We measured the effectiveness of our procedure as follows. (1) We generated an uncorrupted learning sequence; (2) we artificially corrupted this sequence with a positive reading repeated until the end of the sequence (much like Sect. 4.2) and then (3) we applied the procedure. We experimented with $n = 5$ and several corruption rates r: 0, 0.01, 0.05, 0.1, 0.2, 0.35, 0.5. For each learning sequence, S_{check} contained 50 positive readings and 50 negative readings of the page described by the learning sequence. For each pair $\langle r, \text{page} \rangle$, we repeated the test 25 times, with $N_S = 5$ different learning sequences at step 1 and $N_p = 5$ different positive readings at step 2.

Whenever the procedure stated that the learning sequence was corrupted, the test counted as a true positive if $r \neq 0$ and as a false positive otherwise. Whenever the procedure stated that the learning sequence was *not* corrupted, the test counted as a false negative if $r \neq 0$ and as a true negative otherwise.

Figure 5(a) shows the ROC curves, obtained experimenting with different values for m. It can be seen that with TooManyFiringSensors and TooManyFiringInputs the procedure exhibits unsatisfactory performance, in that FPR is never smaller than 0.6 irrespective of m (see also what follows). With TooManyFiringGroups, on the other hand, it appears to exhibit an ideal behavior.

Figure 5(b) plots the positive rate with $m = 1$ as a function of the corruption rate r (the optimum would correspond to a positive rate 0 for $r = 0$ and 1 otherwise). This figure clearly shows that the procedure achieves the optimum (at least in our benchmark) with the TooManyFiringGroups aggregator: it detects each corrupted learning sequence while not misclassifiying any clean sequence. With the two other aggregators, on the other hand, it exhibits far too many false positives. We interpret this result as a consequence of the previous results in Tab. 4: when switching from a clean learning sequence to a corrupted one, the



(a) ROC curves (b) Positive Rate

Fig. 5. Effectiveness of our corruption detection procedure applied to the 3 aggregators. Figure on the left shows ROC curves. Figure on the right plots positive rates for the procedure applied with $m = 1$.

performance change is not sufficiently strong, with TooManyFiringSensors and TooManyFiringInputs.

Another important result from Fig. 5(b) is that the detection accuracy of corrupted sequences is very high over the whole range of r.

7 Concluding Remarks

We attempted to understand the rather unexplored effects of a corrupted learning set in an anomaly-based detection system, with reference to a very specific form of detection. We quantified the impact of the corruption rate on performance indexes (FPR, FNR and A_{ROC}) over many different working points for our detectors. Our experiments confirmed the obvious intuition on the positive correlation between corruption rate and aggregator sensitivity: i.e., as the corruption rate increases the false negative rate (FNR) increases and the false positive rate (FPR) decreases.

We found also the interesting result that a corrupted learning set does not necessarily lead to worse performance, at least for moderate corruption rates. We observed that the threshold leading to best performance depends on the corruption rate (which is unknown, however).

Finally, the previous analysis enabled us to develop a novel *automatic* procedure capable of detecting whether the learning set is corrupted. The procedure differs from other noise detection algorithms in the fact that it does not divide the learning set in smaller sets: this can be an advantage in the cases where learning sets are very small and a further division is not possible. We tested the procedure with our detectors and found that for one of them, the most effective, the procedure exhibits optimal behavior: it is able to detect all the corrupted sets (for each corruption rate value ranging from the minimum to 0.5) while not misclassifying any clean sequence as corrupted.

Acknowledgments

The authors are grateful to the anonymous reviewers and the shepherd for their detailed and constructive comments.

References

1. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, pp. 251–261. ACM Press, New York (2003)
2. Shavlik, J., Shavlik, M.: Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In: KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 276–285. ACM Press, New York (2004)
3. Bartoli, A., Medvet, E.: Automatic Integrity Checks for Remote Web Resources. IEEE Internet Computing 10(6), 56–62 (2006)

4. Bartoli, A., Medvet, E.: Anomaly-based Detection of Web Site Defacements. In submission (2006) Available at `http://www.units.it/~bartolia/abstract/AnomalyBasedDetectionOfWebSiteDefacements.pdf`
5. Lane, T., Brodley, C.E.: An application of machine learning to anomaly detection. In: Proceedings of the Twentieth National Information Systems Security Conference, Gaithersburg, MD, The National Institute of Standards and Technology and the National Computer Security Center, National Institute of Standards and Technology. vol. 1, pp. 366–380 (1997)
6. Lane, T.D.: Machine learning techniques for the computer security domain of anomaly detection. PhD thesis, Purdue University, Major Professor-Carla E. Brodley (2000)
7. Li, K., Teng, G.: Unsupervised svm based on p-kernels for anomaly detection. In: First International Conference on Innovative Computing, Information and Control - vol II (ICICIC'06) 2, pp. 59–62 (2006)
8. Baah, G.K., Gray, A., Harrold, M.J.: On-line anomaly detection of deployed software: a statistical machine learning approach. In: SOQUA '06: Proceedings of the 3rd International Workshop on Software Quality Assurance, pp. 70–77. ACM Press, New York (2006)
9. Zhu, X., Wu, X.: Class noise vs. attribute noise: a quantitative study of their impacts. Artif. Intell. Rev. 22(3), 177–210 (2004)
10. Hodge, V., Austin, J.: A Survey of Outlier Detection Methodologies. Artif. Intell. Rev. 22(2), 85–126 (2004)
11. Brodley, C.E., Friedl, M.A.: Identifying Mislabeled Training Data. J. Artif. Intell. Res (JAIR) 11, 131–167 (1999)
12. Forman, G., Cohen, I.: Learning from little: comparison of classifiers given little training. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) PKDD 2004. LNCS (LNAI), vol. 3202, pp. 161–172. Springer, New York (2004)
13. Hu, W., Liao, Y., Vemuri, V.R.: Robust Support Vector Machines for Anomaly Detection in Computer Security. In: ICMLA, pp. 168–174 (2003)
14. Mahoney, M., Chan, P.: Phad: Packet header anomaly detection for identifying hostile network traffic. Technical report, Florida Tech. CS-2001-4 (2001)
15. Laskov, P., Schäfer, C., Kotenko, I.V.: Intrusion detection in unlabeled data with quarter-sphere Support Vector Machines. In: DIMVA, pp. 71–82 (2004)
16. Tax, D.M., Duin, R.P.: Data Domain Description using Support Vectors. In: ESANN, pp. 251–256 (1999)
17. Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: RAID, pp. 203–222 (2004)
18. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous system call detection. ACM Trans. Inf. Syst. Secur. 9(1), 61–93 (2006)