Expert Systems with Applications 38 (2011) 12521–12530

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Anomaly detection techniques for a web defacement monitoring service

G. Davanzo *, E. Medvet, A. Bartoli

DI3 – Università degli Studi di Trieste, Via Valerio 10, Trieste, Italy

ARTICLE INFO

Keywords:
Security
Web defacement
Machine learning

ABSTRACT

The defacement of web sites has become a widespread problem. Reaction to these incidents is often quite slow and triggered by occasional checks or even feedback from users, because organizations usually lack a systematic and round the clock surveillance of the integrity of their web sites. A more systematic approach is certainly desirable. An attractive option in this respect consists in augmenting availability and performance monitoring services with defacement detection capabilities. Motivated by these considerations, in this paper we assess the performance of several anomaly detection approaches when faced with the problem of detecting web defacements automatically. All these approaches construct a profile of the monitored page automatically, based on machine learning techniques, and raise an alert when the page content does not fit the profile. We assessed their performance in terms of false positives and false negatives on a dataset composed of 300 highly dynamic web pages that we observed for 3 months and includes a set of 320 real defacements.

1. Introduction

Defacements are a common form of attack to web sites. In these attacks the legitimate site content is fully or partly replaced by the attacker so as to include content embarrassing to the site owner, e.g., disturbing images, political messages, forms of signature of the attacker and so on. Defacements are usually carried out by exploiting security vulnerabilities of the web hosting infrastructure, but there is increasing evidence of defacements obtained by means of fraudulent DNS redirections, i.e., by penetrating the DNS registrar rather than the web site. Recent examples of the two strategies include the massive breach suffered by a US hosting company (Hackers hit network solutions customers, 2010) and the redirection that affected a major search site in China (Baidu sues registrar over DNS records hack, 2010). Attackers may focus their efforts toward defacing a specific target site, but often they tend to follow a radically different pattern in which automated tools locate thousands of web sites that exhibit the same vulnerability and can thus be defaced simultaneously, with just a few keystrokes (Danchev, 2008; MultiInjector v0.3 released, 2008).

It seems fair to say that, unfortunately, defacements have gained a sort of first-level citizenship in the Internet. Nearly 1.7 million snapshots of defacements were stored during 2005–2007 at Zone-H, a public web-based archive (http://www.zone-h.org). Back in 2006, the annual survey from the Computer Security Institute observed that "defacement of web sites continues to plague organizations" (Gordon, Loeb, Lucyshyn, & Richardson, 2006). Not only the scenario did not change significantly in the following years, the latest version of this survey reports that the percent of responders which suffered this kind of attack in 2009 more than doubled with respect to 2008—14% vs. 6% (CSI, 2009).

Another side of the problem is the reaction time, i.e., the time it takes to an organization to detect that its site has been defaced and react appropriately. Anecdotal evidence suggests that this is a relevant issue, even at large organizations. To mention just a few examples, the Company Registration Office in Ireland was defaced in December 2006 and remained so until mid-January 2007 (CRO, xxxx). Several web sites of Congressional Members in the house.gov domain were defaced "shortly after President Obama's State of the Union address" and were still defaced at "4:10 am EST" (Congressional web site defacements follow the state of the union, 2010). A systematic study of the reaction time, performed by means of real-time monitoring of more than 60,000 defaced sites extracted on-the-fly from ZoneH, showed that 40% of the defacements in the sample lasted for more than 1 week and 37% of the defacements was still in place after 2 weeks (Bartoli, Davanzo, & Medvet, 2009). The cited study also showed that these figures do not change significantly in sites hosted by Internet providers (and as such presumably associated with systematic administration) nor by taking into account the importance of these sites as quantified by their PageRank index. These data confirm the intuition that web sites often lack a systematic surveillance of their integrity and that the detection of web defacements is usually demanded to occasional checks by administrators or to feedbacks

* Corresponding author.
E-mail addresses: giorgio.davanzo@gmail.com (G. Davanzo), emedvet@units.it (E. Medvet), bartoli.alberto@units.it (A. Bartoli).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.04.038

from users. Indeed, reaction to a defacement occurred recently at Poste.it, one of the largest financial institutions in Italy, was not triggered by site administrators but by a user who called the police because he happened to find the site defaced on Friday late afternoon (Le poste dopo l'attacco web Non violati i dati dei correntisti, 2009). Such an extemporaneous approach is clearly unsatisfactory. A more rigorous and systematic approach capable of ensuring prompt detection of such incidents is required.

An attractive option in this respect consists in augmenting availability and performance monitoring services (e.g., 13 free & cheap website monitoring services, 2008) with defacement detection capabilities (Bartoli & Medvet, 2006; Medvet & Bartoli, 2007). Since these services are cheap and non-intrusive, organizations of essentially any size and budget could indeed afford to exploit these services for performing a systematic and round the clock surveillance against defacements. Indeed, economics seems to play a key role in this scenario. Quantifying the cost of late detection of a defacement is very difficult and weighing this cost against the cost of better security-related skills, practices and technologies is even more difficult. In this respect, an external service that is cheap, can be joined with just a few clicks, without installing any software and without any impact on daily operating activities seems to be a sensible framework for promoting systematic surveillance and quicker detection on a large scale. A service of this kind would also be able to detect defacements induced by fraudulent DNS redirections. Attacks of this form are increasingly more diffused (Baidu sues registrar over DNS records hack, 2010; Google blames DNS insecurity for web site defacements, 2009; Hackers hijack DNS records of high profile new zealand sites, 2009; Puerto rico sites redirected in DNS attack security, 2009) and are very difficult to detect with detection technologies deployed locally on the monitored site.

A crucial problem for successful deployment of a defacement detection service consists in being able to cope with dynamic content without raising an excessive amount of false alarms. Site administrators could provide a description of the legitimate contents for their sites at service subscription time. This option requires defining a site-independent way for collecting this information, whose quality and amount should suffice to cover all relevant portions and content of the monitored pages. Moreover, the option assumes that site administrators indeed have time and skills for actually providing those descriptions. A radically different approach consists in extracting the relevant information automatically by means of *machine learning* techniques. The potential advantages of this approach are obvious, as site administrators would only need to provide the URL of the monitored page and simply wait for a few days—until the service will have constructed a profile of the legitimate content automatically. The implicit assumption is that *anomaly detection* (Denning, 1987; Gosh, 1998; Mutz, Valeur, Vigna, & Kruegel, 2006; Kruegel & Vigna, 2003) is indeed a feasible approach for a monitoring service of this kind, i.e., that defacements indeed constitute anomalies with respect to an established profile of the monitored resource and that false positives may indeed be kept to a minimum despite the highly dynamic nature of web resources.

In this paper we elaborate on this idea and assess the performance of several machine learning approaches when faced with the defacement detection problem. Clearly, by no means we intend to provide an extensive coverage of all the frameworks that could be used (Chandola, Banerjee, & Kumar, 2009; Patcha & Park, 2007; Tsai, Hsu, Lin, & Lin, 2009). We chose to restrict our analysis to key approaches that have been proposed for attack detection at host and network level (Boser, Guyon, & Vapnik, 1992; Breunig, Kriegel, Ng, & Sander, 2000; Kim & Kim, 2006; Lazarevic, Ertoz, Kumar, Ozgur, & Srivastava, 2003; Mukkamala, Janoski, & Sung, 2002; Ramaswamy, Rastogi, & Shim, 2000; Ye, Chen, Emran, & Vilbert, 2000; Ye, Emran, Chen, & Vilbert, 2002; Ye, Li, Chen, Emran, & Xu, 2001; Yeung & Chow, 2002). The analysis includes a detection algorithm that we have developed explicitly for defacement detection and that exploits a fair amount of domain-specific knowledge (Bartoli & Medvet, 2006; Medvet & Bartoli, 2007).

Our evaluation is based on a dataset composed of 300 highly dynamic web pages that we observed periodically for 3 months and on a sample of 320 defacements extracted from ZoneH. Each detection algorithm is hence tested against its ability in not raising false alarms or missing defacements.

2. Our test framework

We developed a prototype framework, which works as follows. We consider a source of information producing a sequence of *readings* $\{r^1, r^2, \ldots\}$ which is input to a *detector*. The source of information is a web page univocally identified by an URL; each reading r consists of the document downloaded from that URL. The detector will classify each reading as being *negative* (legitimate) or *positive* (anomalous). The detector consists internally of a *refiner* followed by an *aggregator*, as represented in Fig. 1.

2.1. Refiner

The refiner implements a function that takes a reading r and produces a fixed size numeric vector $v \in \mathbb{R}^n$. The refiner is internally composed by a number of *sensors*. A sensor is a component which receives as input a reading and outputs a fixed size vector of real numbers. The output of the refiner is obtained by concatenating outputs from the 43 different sensors of our prototype and corresponds to a vector of 1466 elements (Medvet & Bartoli, 2007). Sensors are functional blocks and have no internal state: v depends only on the current input r and does not depend on any prior reading.

Sensors are divided in five categories, accordingly to the way in which they work internally. Table 1 indicates the number of sensors and the corresponding size for the vector v portion in each category.

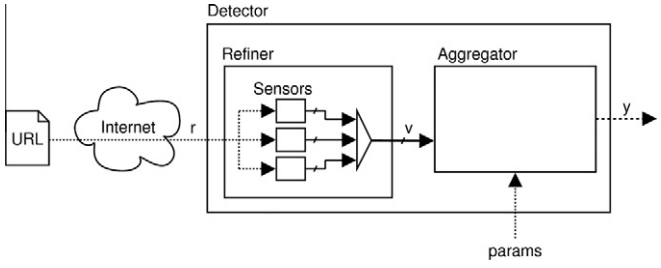

Fig. 1. Detector architecture. Different arrow types correspond to different types of data.

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

2.1.1. 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).

2.1.2. RelativeFrequencies sensors

Each sensor s in this category outputs a vector composed by n_s elements $v = |v^1, \ldots, v^{n_s}|$. 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 kth 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 $n_s = 91$.

2.1.3. HashedItemsCounter sensors

Each sensor s in this category outputs a vector composed by n_s elements $v = |v^1, \ldots, v^{n_s}|$ and works as follows. Given a reading i, s: (1) sets to 0 each element v^k of v; (2) builds a set $L = \{l_1, l_2, \ldots\}$ 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 $1 \leqslant k_j \leqslant n_s$; (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 five different sub-features, i.e., by five 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.

2.1.4. HashedTree sensors

Each sensor s in this category outputs a vector composed by n_s elements $v = |v^1, \ldots, v^{n_s}|$ 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 sensors 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.

2.1.5. 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 four 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.

2.2. Aggregator

The aggregator is the core component of the detector and it is the one that actually implements the anomaly detection. The aggregator output y can be one of the following: `negative`, if the input r is classified as legitimate, and `positive`, if the reading is classified as anomalous.

The aggregator internally uses a profile to classify readings, which is constructed before starting the monitoring activity. The aggregator compares the current reading against the profile and classifies it as anomalous according to an aggregator-specific criterion. The profile is computed using a *learning sequence* S composed by genuine readings (that thus have to be classified as negative) collected by observing the web page during a preliminary *learning phase* and sample attacks provided by an operator (that thus have to be classified as positive). We will denote by S^- and S^+ the portions of S containing genuine readings and attacks, respectively.

As will be described in more detail when discussing the experiments, the learning sequence for a resource contains only genuine readings of that resource (beyond the sample attacks). In other words, we constructed a different profile for each resource. In principle, one could build one single profile attempting to capture all resources that are not defacements and then classify as anomalous any deviation from that profile. We did not pursue this option because in preliminary experiments, not reported here for brevity, we could not devise any clear and sharp separation among legitimate pages and defacements.

In the comparative evaluation of techniques presented in the next sections, we set an aggregator that we developed earlier as baseline (Bartoli & Medvet, 2006). This aggregator implements a form of anomaly detection based on *Domain Knowledge* and is shortly described in Section 3.6.

3. Anomaly detection techniques

In this section we describe the techniques that we assessed in this comparative experimental evaluation: all these techniques

have been proposed and evaluated for detecting intrusions in host or network based IDSs. We chose not to include any forms of Bayes classifier because of the difficulty in selecting meaningful values for the prior probability of suffering a defacement. While such estimates may be obtained easily for network-level attacks or for spam filtering (in those cases any organization exposed on the Internet will collect a sufficient amount of positive samples in a short time), we could not devise any practical way for obtaining values meaningful in our scenario.

Each technique consists of an algorithm aimed at producing a binary classification of an item expressed as a numeric vector (or point) $p \in \mathbb{R}^n$. We incorporated each technique in our framework by implementing a suitable aggregator, that is, all these techniques are based on the same refiner and apply different criteria for classifying a reading. Each technique constructs a profile with a technique-specific method, basing on the readings contained in a learning sequence S. The learning sequence is composed by both negative readings (S^-) and positive readings (S^+). Only one of the techniques analyzed (Support Vector Machines) indeed make use of S^+, however. For all the other methods $S = S^-$ and $S^+ = \emptyset$.

3.1. kth nearest

This technique (Kim & Kim, 2006; Lazarevic et al., 2003; Ramaswamy et al., 2000) is distance-based, often computed using Euclidean metric. For this aggregator the profile consists of the learning sequence S^-. Let k be an integer positive number and p the investigated point; we define the kth nearest distance $D^k(p)$ as the distance $d(p,o)$ where o is a generic point of S^- such that:

1. for at least k points $o' \in S^-$ it holds that $d(p,o') \leqslant d(p,o)$, and
2. for at most $k-1$ points $o' \in S^-$ it holds that $d(p,o') < d(p,o)$.

We define a point p as a positive if $D^k(p)$ is greater than a provided threshold t.

In our experiments we used the Euclidean distance, we set $k = 3$ and $t = 1.01$.

3.2. Local Outlier Factor

Local Outlier Factor (from here on LOF) (Breunig et al., 2000; Lazarevic et al., 2003) is an extension to the kth nearest distance, assigning to each evaluated point an *outlying degree*. The main advantage of LOF on other distance based techniques is the way it handles the issue of varying densities in the data set, which is especially useful when points of a class are scattered so as to form different clusters. The LOF value of a point p represents its outlying degree computed as the ratio of the average local density of the k nearest neighbors and the local density of p, as follows. Also in this case, the profile simply contains the learning sequence S^-.

1. compute the kth nearest distance $D^k(p)$ and define *k-distance neighborhood* $N_k(p)$ as a set containing all the points $o \in S^-$ such that $d(p,o) \leqslant D^k(p)$;
2. define reachability distance reach-dist$(o,p) = \max\{D^k(p), d(o,p)\}$;
3. compute local reachability density lrd (p) as the inverse of the average reachability distance of points belonging to $N_k(p)$;
4. finally, LOF value $LOF(p)$ is defined as the average of the ratios of lrd(o), with $o \in N_k(p)$, and lrd(p).

A point p is defined as a positive if $LOF(p) \notin \left[\frac{1}{1+\epsilon}, 1+\epsilon\right]$, where ϵ represents a threshold. In our experiments we used the Euclidean distance for d, setting $\epsilon = 1.5$.

3.3. Hotelling's T-Square

Hotelling's T-Square method is a test statistic measure for detecting whether an observation follows a multivariate normal distribution. It has been proposed for intrusion detection on the ground that when the observed variables are independent and their number is sufficiently large (a few tens), then the T-Square statistics of the observations follows approximately a normal distribution irrespective of the actual distribution of each variable (Hotelling, 1931; Ye et al., 2000, 2001, 2002). This technique is based on the covariance matrix C composed by all elements of S^- and by the investigated point p. The profile consists of C and of the averages vector μ computed on the learning sequence S^-.

Hotelling's T-Square statistic is defined as:

$$t^2(p) = m(p-\mu)^T C^{-1}(p-\mu)$$

where m is the length of S^- and μ is the vector of the averages of S^- vectors. If C is a singular matrix, we slightly modify it until it becomes non-singular by adding a small value to its diagonal.

We define a point p as positive if $t^2(p) > \max\{t^2(o) \mid o \in S^-\} + t$, where t is a predefined threshold. In our experiments we set $t = 5$.

This method is very similar to the one used in Mahalanobis (1936) and Lazarevic et al. (2003), based on the Mahalanobis distance; we actually implemented both aggregators, but since the results are almost identical we will further investigate only the one based on Hotelling's T-Square.

3.4. Parzen windows

Parzen Windows (Kim & Kim, 2006; Parzen, 1962; Yeung & Chow, 2002) provide a method to estimate the probability density function of a random variable. The profile consists of the learning sequence S^-.

Let $p = \{x_1, x_2, \ldots, x_n\} \in R^n$ be the investigated point and let X_i be a random variable representing the ith component of p. We need to approximate the unknown density function $f(x_i)$ of each X_i. Having obtained such an approximation $\tilde{f}(x_i)$, as described below, we will say that a component x_i of p is anomalous if and only if $\tilde{f}(x_i) < t_1$ and we will classify point p as positive if a percentage of its components greater than t_2 is anomalous (t_1 and t_2 being two parameters). In other words, with this method a probability distribution is estimated for each component of the input vector using its values along the learning sequence; then an alarm is raised if too many components seem not to agree with their estimated distribution.

Let the *window function* $w(x)$ be a density function such that its volume is $V_0 = \int_{-\infty}^{+\infty} f(x)dx$. We considered two window functions:

$$\text{Gaussian}: \quad w(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

$$\text{Pulse}: \quad w(x) = \begin{cases} 1 & \text{if } -\alpha \leqslant x \leqslant \alpha \\ 0 & \text{otherwise} \end{cases}$$

We approximate $f(x_i)$ as follows (x_i^k is the value of the ith component of the kth point of S^-):

$$\tilde{f}(x_i) = \frac{1}{n}\sum_{k=1}^{n} \frac{1}{V_k} w\left(\frac{x_i - x_i^k}{V_k}\right) \tag{1}$$

where $V_k = \frac{V_0}{\ln k}$; note that the weight term V_k decrease for older values (i.e., for points of S^- with higher k).

We set $\sigma = 1$, $t_1 = 0.1$ and $t_2 = 7.5\%$ for Parzen Gaussian and $\alpha = 0.25$, $t_1 = 0.3$ and $t_2 = 10\%$ for Parzen Pulse.

3.5. Support Vector Machines

Support Vector Machines (SVM) (Boser et al., 1992; Lazarevic et al., 2003; Mukkamala et al., 2002) use hyperplanes to maximally

separate N classes of data, where $N = 2$ in our setting. This technique uses a kernel function to compute the hyperplanes using both readings of S^- and S^+. In our experiments we used the *Radial Basis Function* (as part of the `libsvm` implementation (Chang & Lin, 2001)).

Once the hyperplanes are defined, a point p is considered as a positive if it is contained in the corresponding class. The profile stores the support vectors computed on the whole learning sequence S, i.e., both S^- and S^+.

3.6. Domain Knowledge aggregator

This aggregator exploits the knowledge about the refiner structure and hence about the meaning of the sensor associated with each slice of v. In a nutshell, this aggregator transforms each slice in a boolean value by applying a sensor-specific transformation. The transformation involves a comparison against a sensor-specific profile computed in the learning phase. When the boolean obtained from a slice is true, we say that the corresponding sensor *fires*. If the number of categories with at least one sensor that fires is at least t ($t = 3$ in our experiments), the reading is classified as anomalous.

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

3.6.1. Cardinality

In the learning procedure the aggregator determines mean η and standard deviation σ of the values $\{v_1^1, \ldots, v_l^1\}$—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 $|v^1 - \eta| \geqslant 3\sigma$.

3.6.2. 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 $\{\eta_1, \ldots, \eta_{n_s}\}$ of the vector elements associated with s; (ii) computes the following for each reading v_k of the learning sequence ($k \in [1,l]$):

$$d_k = \sum_{i=1}^{n_s} |v_k^i - \eta_i| \tag{2}$$

(iii) computes mean η and standard deviation σ of $\{d_1, \ldots, d_l\}$.

In the monitoring phase, for a given reading v, the aggregator computes:

$$d = \sum_{i=1}^{n_s} |v^i - \eta_i| \tag{3}$$

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

3.6.3. 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. $\{m_1, \ldots, m_{n_s}\}$. 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 $\{m_1, \ldots, m_{n_s}\}$, 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.

3.6.4. 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 $\{m_1, \ldots, m_{n_s}\}$ 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.

3.6.5. 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.

4. Experiments and results

4.1. Datasets

We built two datasets different both in time span and number of resources. The *Large–Long dataset* consists of snapshots of 300 highly dynamic web resources[1] that we sample every 6 h for 3 months, thus totaling a *negative sequence* of 350 readings for each web page web pages for 3 months, collecting a reading for each page every 6 h, thus totaling a *negative sequence* of 350 readings for each web page. The archive include technical web sites (e.g., The Server Side, Java Top 25 Bugs), newspapers and news agencies both from the US and from Italy (e.g., CNN Business, CNN Home, La Repubblica, Il Corriere della Sera), e-commerce sites (e.g. Amazon Home), sites of the Italian public administration, the top 100 blogs from CNET, the top 50 Universities and so on. Some resources were hand picked (mostly those in the technicals, e-commerce and USA newspapers groups) while the others were collected automatically by selecting the most popular resources from public online lists (e.g.: `topuniversities.com` for Universities). Almost all resources contain dynamic portions that change whenever the resource is accessed. In most cases such portions are generated in a way hardly predictable (including advertisements) and in some cases they account for a significant fraction of the overall content.

We also collected an *attack archive* composed by 320 readings extracted from a publicly available defacements archive (http://www.zone-h.org). The set is composed by a selection of real defacements performed by different hackers or teams of hackers: we chose samples with different size, language, layout and with or without images, scripts and other rich features. We attempted to build a set with wide coverage and sufficiently representative of real-world defacements.[2]

The *Small–Short dataset* is the same that we used in our previous works (Medvet & Bartoli, 2007; Bartoli & Medvet, 2006). It is a subset of the previous dataset both in number of resources and in time length. It is composed by 15 resources (listed in Table 3) limited to 125 contiguous snapshots. The attack archive is composed by 95 readings of the previous attack archive (selected so as to be representative enough of the full sample).

[1] For a complete list, please visit http://tinyurl.com/exsa-resources-1.
[2] For a complete list, please visit http://tinyurl.com/exsa-attacks-1.

Table 2
Time span for the two datasets.

Dataset	Number of resources	S^-		S^+	S_t^-		S_t^+
		#	Days	#	#	Days	#
Small–Short	15	50	12	20	75	19	75
Large–Long	300	50	12	20	300	75	300

Table 3
List of web pages composing the Small–Short dataset. Concerning Amazon – Home page and Wikipedia – Random page, we noticed that most of the content section of the page changed at every reading, independently from the time.

Web page
Amazon – Home page
Ansa – Home page
Ansa – Rss sport
ASF France – Home page
ASF France – Traffic page
Cnn – Businnes
Cnn – Home page
Cnn – Weather
Java – Top 25 bugs
Repubblica – Home page
Repubblica – Tech. and science
The Server Side – Home page
The Server Side – Tech talks
Univ. of Trieste – Home page
Wikipedia – Random page

4.2. Methodology

We used False Positive Ratio (FPR) and False Negative Ratio (FNR) as performance indexes. For each page we executed a learning phase followed by a testing phase in which we measured false positives and false negatives. Table 2 summarizes the length and corresponding time frames for the learning and testing sequences, as explained in detail below.

In detail, in the learning phase: (i) we built a sequence S^+ of positive readings composed by 20 random elements of the attack archive; (ii) we built a sequence S^- of negative readings composed by the first 50 elements of the corresponding negative sequence (i.e., roughly 2 weeks); (iii) we built the learning sequence S by joining S^+ and S^-; and (iv) we trained each aggregator on S (recall that only one aggregator actually looks at S^+, as pointed out in Section 3).

In the monitoring phase: (i) we built a testing sequence S_t by joining a sequence S_t^-, composed by the remaining readings of the corresponding negative sequence (300 or 75, depending on the dataset), and a sequence S_t^+, composed by the remaining readings of the attack archive (again, 300 or 75 up to the used dataset); (ii) we fed the aggregator with each reading of S_t. We counted each anomaly raised for elements of S_t^- as a false positive. We counted each element of S_t^+ not raised as an anomaly as a false negative.

4.3. Preliminary results

This experiment has been conducted on the Small–Short dataset. The results are given in Table 4.

Concerning defacement detection, it can be seen that 5 out of 7 aggregators managed to detect every injected defacement (i.e., FNR = 0%). SVM and LOF heavily failed, scoring an average FNR of 30% and 11% respectively.

In terms of false positives, DomainKnowledge provided the best result with FPR below 4%. The next best value is that of LOF, whose FNR is unsatisfactory. FPR results indicate that the very good FNR results for KNearest and Hotelling actually are misleading: these two aggregators exhibit a strong tendency to classify every element in S_t as being anomalous, irrespective of its actual content with respect to the profile.

Table 4
Preliminary results on the Small–Short dataset.

Aggregator	FPR %			FNR %		
	AVG	MAX	StdDev	AVG	MAX	StdDev
KNearest	100.00	100.00	0.00	0.00	0.00	0.00
SVM	27.56	96.00	38.90	29.93	57.33	26.89
PulseParzen	6.44	84.00	19.15	0.00	0.00	0.00
GaussianParzen	11.70	84.00	19.93	0.00	0.00	0.00
DomainKnowledge	3.56	53.33	12.16	0.00	0.00	0.00
LOF	3.63	49.33	11.37	11.33	100.00	31.36
Hotelling	96.59	100.00	7.33	0.00	0.00	0.00

4.4. Feature selection

Despite the fact that the previous experiment was carried out on the Small–Short dataset, its execution required a large amount of time and computing resources. Based on this observation, coupled with the unsatisfactory performance of all aggregators (except for DomainKnowledge), we decided to execute further experiments by applying a *dimensionality reduction* to the dataset. Clearly, this choice adds a further dimension to the design space, due to the many techniques that could be used (Ailon & Chazelle, 2010; Jolliffe, 2002; Kriegel, Krger, & Zimek, 2009; Tsai et al., 2009). We chose to restrict our analysis to a *feature selection* procedure based on a backward elimination algorithm similar to the one proposed in Song, Smola, Gretton, Borgwardt, and Bedo (2007) and detailed below. For each resource we selected a subvector of v including only its elements which appear to have more significance in the decision, i.e., those with maximal correlation with the desired output.

All the experiments in the following sections have been performed with feature selection enabled for all the aggregators except for the DomainKnowledge. Since its performance and runtime cost do not make the use of the full vector v unfeasible, we thought that exercising this aggregator with less features than it may handle would not be very interesting.

The feature selection algorithm is applied once for each web page and works as follows. Let X_i denote the random variable associated with the i-th element of v (i.e., v_i) across all readings of S. Let Y be the random variable describing the desired values for the aggregator: $Y = 0$, $\forall$ reading $\in S^-$; $Y = 1$ otherwise, i.e., $\forall$ reading $\in S^+$.

We computed the absolute correlation c_i of each X_i with Y and, for each pair $\langle X_i, X_j \rangle$, the absolute correlation $c_{i,j}$ between X_i and X_j. Then, we executed the following iterative procedure, starting from a set of unselected indexes $I_U = \{1, \ldots, 1466\}$ and an empty set of selected indexes $I_S = \emptyset$: (1) we selected the element $i \in I_U$ with the greatest c_i and moved it from I_U to I_S; (2) $\forall j \in I_U$, we set $c_j := c_j - c_{i,j}$. We repeated these two steps until a predefined size s for I_S is reached. We selected for each technique the maximum value of s that appeared to deliver acceptable performance: we set $s = 10$ for kth nearest, LOF and Hotelling and $s = 20$ for the others. Point p will include only those elements of vector v whose indexes are in I_S. In other words, we take into account only those indexes with maximal correlation with the desired output (step 1), attempting to filter out any redundant information (step 2).

4.5. Results with feature selection

Table 5 shows the results for the Small–Short dataset, which are much better than those obtained on the same dataset without fea-

Table 5
Small–Short dataset: results with feature selection.

Aggregator	FPR %			FNR %		
	AVG	MAX	StdDev	AVG	MAX	StdDev
KNearest	0.52	6.67	1.61	0.30	4.00	0.95
SVM	0.00	0.00	0.00	0.00	0.00	0.00
PulseParzen	0.30	4.00	0.95	0.00	0.00	0.00
GaussianParzen	10.81	69.33	19.32	0.00	0.00	0.00
DomainKnowledge	3.56	53.33	12.16	0.00	0.00	0.00
LOF	5.41	49.33	12.56	0.00	0.00	0.00
Hotelling	5.63	53.33	13.34	0.00	0.00	0.00

Table 6
Large–Long dataset: results with feature selection.

Aggregator	FPR %			FNR %		
	AVG	MAX	StdDev	AVG	MAX	StdDev
KNearest	19.43	100.00	29.94	0.85	79.67	6.24
SVM	6.45	100.00	17.80	0.10	15.33	0.91
PulseParzen	14.66	100.00	25.07	0.20	8.00	0.74
GaussianParzen	28.48	100.00	31.99	0.08	5.33	0.54
DomainKnowledge	19.10	100.00	30.42	0.02	5.67	0.33
LOF	24.18	100.00	33.95	6.21	99.00	20.15
Hotelling	24.76	100.00	32.33	0.27	26.00	1.60

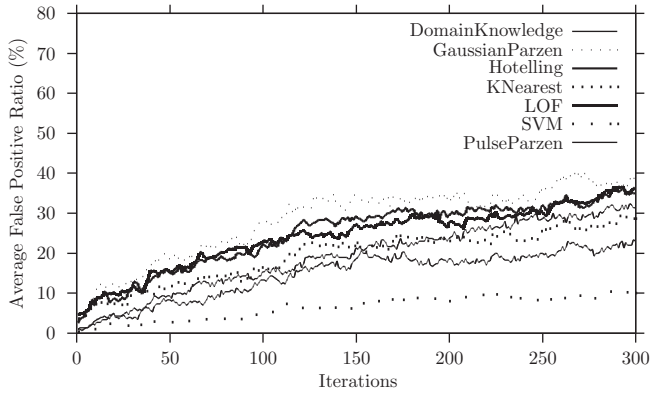

Fig. 2. False Positive Ratio over time (expressed as index n in the negative samples of the testing sequence S_t^-).

ture selection (Table 4; recall that the result of DomainKnowledge are identical in the two scenarios).

Concerning defacement detection, FNR values suggest that all the techniques proved to be effective when detecting defacements. From the point of view of false positives, the behavior of all the aggregators improve but their performance is not equivalent.

An excellent result comes from SVM, which managed to classify all the negative readings correctly, while still detecting all the attacks. LOF and Hotelling managed to score as well as the DomainKnowledge, while KNearest and PulseParzen scored even better; GaussianParzen did not score well, being unable to classify genuine pages for several resources.

Table 6 shows the results for the Large–Long dataset. Performance with this larger and longer sample are sensibly worse than in the previous scenario, from all points of view. SVM still exhibits the best results.

A deeper analysis of the raw data suggests that the main reason for the worse performance could be the increase in the time interval used for validation rather than the larger sample size. Fig. 2 shows the average FPR at every time instant, expressed as the index n of the reading of S_t^-. It can be observed that many aggregators perform well at the beginning but then FPR steadily increases for all the aggregators, reaching unacceptable values quickly. From a different point of view, the quality of the profile constructed in the learning phase degrades progressively and eventually becomes no longer adequate. It seems mandatory, thus, to adopt an approach in which the profile of the resources is upgraded at regular intervals.

4.6. Results with feature selection and retuning

The previous experiments were done by constructing the profile of each web page only once, that is, by locking the internal state of each aggregator.

We also experimented without locking the internal state, as follows. After the initial learning phase, whenever a reading of S_t^- was evaluated, we added that reading to S^- and removed the oldest reading from S^- (in other words, we used a sliding window of the 50 most recent readings of S^-); then, a new profile was immediately computed using the updated S^- (S^+ is never changed). In this way, we enabled a sort of continuous *retuning* that allowed each aggregator to keep the corresponding profile in sync with the web page, even for long time frames (i.e., the large dataset).

Please note that we froze the aggregator state before submitting positive samples, since we are not interested into measuring the system effectiveness when detecting several different defacements after a false negative; hence, the results in terms of FNR will be the same of Section 4.5.

Table 7 shows the results we obtained. As expected, all techniques exhibited a sensibly lower FPR; nevertheless, some aggregators (KNearest, PulseParzen and GaussianParzen) still show a high value of maximum FPR.

The SVM aggregator scored even better than Domain Knowledge in terms of FPR (0.07% vs. 0.25%); however, the Domain Knowledge still proves to be the best one when comparing FNRs (0.10% vs. 0.02%).

4.7. Elaboration time

A key element of the system hereby proposed is its capability to provide a result in a reasonably low amount of time. Table 8 shows

Table 7
Large–Long dataset: results with feature selection and retuning.

Aggregator	FPR %			FNR %		
	AVG	MAX	StdDev	AVG	MAX	StdDev
KNearest	2.23	38.67	5.50	0.85	79.67	6.24
SVM	0.07	2.00	0.19	0.10	15.33	0.91
PulseParzen	0.46	26.00	1.67	0.20	8.00	0.74
GaussianParzen	2.28	50.00	4.47	0.08	5.33	0.54
DomainKnowledge	0.25	4.33	0.47	0.02	5.67	0.33
LOF	3.18	15.33	3.68	6.21	99.00	20.15
Hotelling	0.74	5.00	0.88	0.27	26.00	1.60

Table 8
Average computation time of a single resource snapshot for different policies (ms).

Aggregator	Retuning: NO FeatSel: NO	Retuning: NO FeatSel: YES	Retuning: YES FeatSel: YES
KNearest	70.24	3.87	3.84
SVM	41.93	3.97	3.93
PulseParzen	3.36	3.94	4.07
GaussianParzen	6.02	4.02	3.46
DomainKnowledge	0.18	0.18	0.82
LOF	111.87	5.10	4.95
Hotelling	13175.68	3.12	3.46

average elaboration times for different system configurations (as explained in the three previous sections) expressed in milliseconds. All the computations were performed on a twin quad core Intel Xeon E5410 with 8 Gb of ram; please note that notwithstanding the elevate number of cores, the evaluation of every resource snapshot is always contained in a single thread—thus using a single core.

The Domain Knowledge is the fastest in all the configurations; as expected, its response time slightly increases (+70 ms) when the continuous retuning is performed.

All the other aggregators are slower by at least three orders of magnitude when not selecting the features (i.e., when all the 1466 elements of the vector are considered), with Hotelling requiring almost 13 s for every iteration.

On the other side, performing the continuous retuning does not require too much additional time for the not-Domain Knowledge aggregators.

Table 9 shows how scalable each aggragator is, measured as snapshots that each CPU core could examine during 1 h (not considering delays on the network-end of the system).

Being the fastest, the DomainKnowledge could process more than 70,000 snapshots per hour on a single-core CPU; all the other aggregators lay in the 12,000—15,000 interval. Since each aggregator is trained for every resource, different resources can be observed in different threads, thus allowing an easy way to scale on several CPU cores/machines.

Table 9
Resource snapshots evaluated per hour on a single core.

Aggregator	Snapshots/h
KNearest	15,645
SVM	15,263
PulseParzen	14,749
GaussianParzen	17,356
DomainKnowledge	73,620
LOF	12,111
Hotelling	17,321

4.8. Discussion of feature selection

In this section we will investigate how the feature selection algorithm acts on the Large–Long dataset. As described in Section 4.4, we execute the feature selection algorithm once for each resource. The algorithm takes into account only features with maximal correlation with the desired output, attempting to filter out any redundant features.

Fig. 3 plots the number of times each feature has been selected. Features are sorted in decreasing order for the sake of clarity. It can be seen that the selection count is highly skewed: only 350 features over 1466 have been selected at least once; less than 30 features were selected in more than 20% resources; only four features have been used in more than half of the 300 resources.

Fig. 4 provides a different view of the data, at the sensor level rather than at the feature level. The figure plots how many times, in percentage over the full dataset, each sensor has been used. We say that a sensor is used when at least one of its features is selected (recall that the number of features associated with each sensor depends on the nature of the sensor itself, in Table 1). It can be seen that usage data is highly skewed also at the sensor level: 10 out of 43 sensors are never selected. Interestingly, 9 of the never

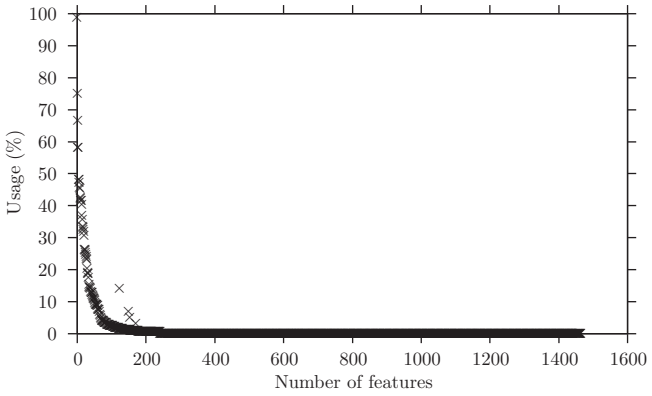

Fig. 3. Feature selection count.

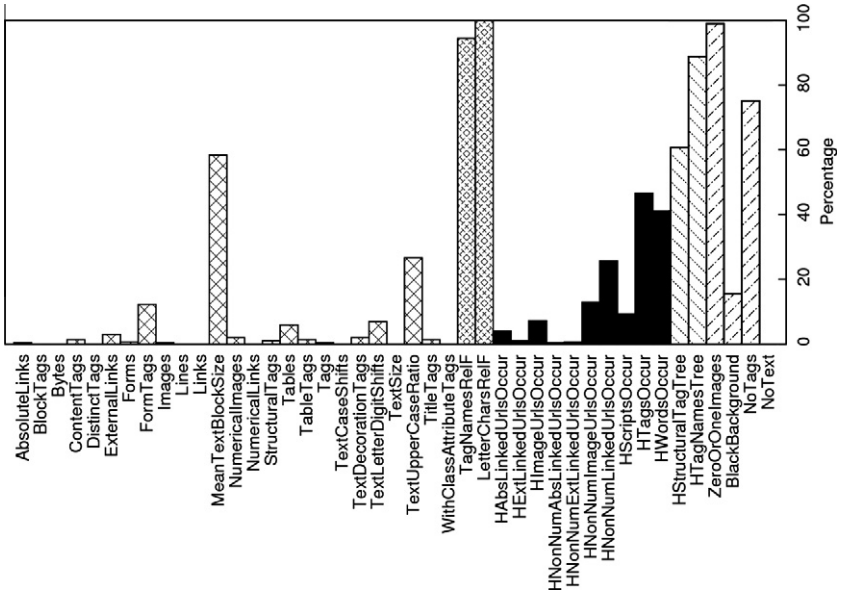

Fig. 4. Sensors usage. Different patterns relate to different categories.

used sensors belongs to the Cardinality category, which means all these sensors are associated with only one feature. Sensors in the RelativeFrequencies categories are used almost always, whereas HashedTree and Signature sensors are also used quite often.

Although feature selection turns out to be a necessity (except for the DomainKnowledge aggregator) a question arises whether this choice might facilitate the attackers' job. In particular, an attacker could choose to modify only features that are not selected by the feature selection algorithm. For example, the attacker could target a never used sensor like TextCaseShift, i.e., by altering all the page text elements so as to introduce a continuous upper to lower case shift. This attack strategy would circumvent completely all the aggregators (again, except for DomainKnowledge).

In other words, automated defacement detection could become an adversarial game and use of feature selection would be an intrinsic potential weakness of the defense strategy. Whether this potential weakness may or may not be practically relevant is unclear, though. Along the line of the previous example, altering the text case distribution in a page could be a satisfactory objective for some attackers (and an embarrassing breach for some defenders) but not for others. Indeed, whether such a change would match the notion of defacement is questionable. Moreover, there is in general a functional dependency between features, in the sense that user-interesting properties of page are generally reflected on several features. For example, an attacker could target other unused sensors like the NoText, Bytes and Links and consequently remove all the text from a page. On the other hand, such a change would affect other sensors including some of those that tend to be selected often like TagNamesRelativeFrequency and alike. It follows that crafting defacements so as to systematically focus on features irrelevant for the aggregator may or may not be an effective attack strategy and further research is required in this respect.

We also observe that the attacker might not have precise knowledge of the set of selected features because the outcome of the feature selection algorithm depends on the composition of the attack set, which may be varied and/or kept secret. Indeed, in detection systems based on forms of machine learning—like ours—the standard security practice consists in assuming that the learning algorithm is common knowledge whereas the set of features used by a specific instance of the algorithm is kept secret, which is feasible in systems with a small number of deployments (Barreno, Nelson, Sears, Joseph, & Tygar, 2006). A materialization of this principle can be found in spam filtering: while the general principles of spam filtering tools are well known, the exact details and settings of the tools used, say, by Gmail or Hotmail, are kept secret.

5. Concluding remarks

We assessed experimentally the performance of several techniques for automated anomaly-based detection of web defacements. According to our results, DomainKnowledge, SVM, PulseParzen and Hotelling all exhibit FNR and FPR values sufficiently low to deserve consideration—average FPR lower than 1%, while being able to correctly detect almost all the simulated attacks (FNR $\simeq$ 0%). Such a finding combined with the moderate computing cost, in particular for DomainKnowledge, suggests that the approach may be practical. KNearest, GaussianParzen and LOF, on the other hand, do not appear to provide adequate performance. It could be interesting to perform a deeper analysis of the impact of size and quality of the attack set on the performance of SVM, being the only approach whose training requires positive samples.

A feature selection aimed at drastically reducing the dimension of the input space turned out to be a necessity for all the approaches, except for DomainKnowledge, both from the point of view of performance and computing cost. In an adversarial scenario, the fact that an aggregator systematically ignores certain features might constitute an opportunity worth exploring by the attackers.

Our DomainKnowledge aggregator is one of those that exhibits better performance and appears to have two key advantages over the other alternatives. First, it is intrinsically able to provide an explanation for the alerts. It suffices, for example, to associate each alert with a summary of the sensors that fired—e.g., an anomalous number of links in the page, a missing tag and alike. The ability to understand the reason for an alert easily is often deemed essential by operators (Xu, Huang, Fox, Patterson, & Jordan, 2009) and may allow detecting false positives more quickly. These indications can hardly be provided using the other techniques. Second, it allows exploiting apriori knowledge about the monitored resources, when available and appropriate. An actual deployment of a service based on DomainKnowledge aggregator could allow administrators of monitored site to declare that the firing of a certain sensor is a sufficient condition for generating an alert. For example, the banner of a site might be a component that should never change—a conclusion that may not be drawn automatically based by merely observing the training set. Providing similar functionality with the other techniques appear to be quite difficult.

Acknowledgement

A preliminary short version of this work can be found in the Proceedings of the IFIP Tc 11 23rd International Information Security Conference 711–716 (2008), at http://dx.doi.org/10.1007/978-0-387-09699-5_50.

References

13 free and cheap website monitoring services. URL: <http://mashable.com/2008/08/25/free-and-cheap-website-monitoring-services/>

Ailon, N., & Chazelle, B. (2010). Faster dimension reduction. *Communications of the ACM, 53*(2), 97–104. doi:10.1145/1646353.1646379. URL: <http://portal.acm.org/ft_gateway.cfm?id=1646379&type=html&coll=portal&dl=ACM&CFID=75438700&CFTOKEN=90489670>.

Baidu sues registrar over DNS records hack (2010). URL: <http://www.theregister.co.uk/2010/01/20/baidu_dns_hack_lawsuit>.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2006). Can machine learning be secure?In *Proceedings of the 2006 ACM symposium on information, computer and communications security* (pp. 16–25). Taipei, Taiwan; ACM. Invited talk. doi:10.1145/1128817.1128824. URL: <http://portal.acm.org/citation.cfm?id=1128817.1128824>.

Bartoli, A., Davanzo, G., & Medvet, E. (2009). The reaction time to web site defacements. *IEEE Internet Computing, 13*(4), 52–58.

Bartoli, A., & Medvet, E. (2006). Automatic integrity checks for remote web resources. *IEEE Internet Computing, 10*, 56–62.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Annual workshop on computational learning theory* (pp. 144–152). Pittsburgh, Pennsylvania, United States: ACM.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: identifying density-based local outliers. *SIGMOD Record, 29*, 93–104.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys, 41*(3), 1–58. doi:10.1145/1541880.1541882.

Chang, C. C., & Lin, C. J. (2001). Libsvm: A library for support vector machines (Vol. 80, pp. 604–611). Software available at <http://www.csie.ntu.edu.tw/cjlin/libsvm>.

Congressional web site defacements follow the state of the union (2010). URL: <http://praetorianprefect.com/archives/2010/01/congressional-web-site-defacements-follow-the-state-of-the-union/>.

CRO website hacked. URL: <http://www.siliconrepublic.com/news/news.nv?storyid=single7819>.

CSI (2009). 14th annual CSI computer crime and security survey – executive summary, Technical report, Computer Security Institute.

Danchev, D. (2008). A commercial web site defacement tool (April 2008). URL: <http://ddanchev.blogspot.com/2008/04/commercial-web-site-defacement-tool.html>

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering, 13*, 222–232.

Google blames DNS insecurity for web site defacements (May 2009). URL: <http://www.infoworld.com/t/authentication-and-authorization/google-blames-dns-insecurity-web-site-defacements-722>

Gordon, L., Loeb, M., Lucyshyn, W., & Richardson, R. (2006). 11th annual CSI/FBI computer crime and security survey, Technical report, Computer Security Institute.

Gosh, A. K. (1998). Detecting anomalous and unknown intrusions against programs. In *14th Annual Computer Security Applications Conference*.

Hackers hijack DNS records of high profile new zealand sites (April 2009). URL <http://blogs.zdnet.com/security/?p=3185>

Hackers hit network solutions customers (2010). URL: <http://www.pcworld.com/businesscenter/article/187250/hackers_hit_network_solutions_customers.html>.

Hotelling, H. (1931). The generalization of student's ratio. *The Annals of Mathematical Statistics, 2*, 360–378.

Jolliffe (2002). *Principal component analysis*. Springer.

Kim, E., & Kim, S. (2006). Anomaly detection in network security based on nonparametric techniques. In *Proceedings of the INFOCOM 2006. 25th IEEE international conference on computer communications* (pp. 1–2).

Kriegel, H., Krger, P., & Zimek, A. (2009). Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery Data, 3*(1), 1–58. doi:10.1145/1497577.1497578. URL <http://portal.acm.org/citation.cfm?id=1497577.1497578& coll=ACM&dl=ACM&CFID=45877002&CFTOKEN=68158520>.

Kruegel, C., & Vigna, G. (2003). Anomaly detection of web-based attacks. In *Conference on computer and communications security* (pp. 251–261). Washington, DC, USA: ACM.

Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003). A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the third SIAM international conference on data mining*.

Le poste dopo l'attacco web Non violati i dati dei correntisti, Repubblica.it. URL: <http://www.repubblica.it/2009/10/sezioni/tecnologia/poste-attacco/dati-non-rischiano/dati-non-rischiano.html>.

Mahalanobis, P. C. (1936). On the generalized distance in statistics. In *Proceedings of the National Institute of Science of India* (Vol. 12, pp. 49–55).

Medvet, E., & Bartoli, A. (2007). On the effects of learning set corruption in anomaly-based detection of web defacements. Detection of intrusions and malware, and vulnerability assessment. URL: <http://dx.doi.org/10.1007/978-3-540-73614-1_4>

Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In *IJCNN '02. Proceedings of the 2002 international joint conference on neural networks* (Vol. 2, pp. 1702–1707).

MultiInjector v0.3 released (November 2008). URL: <http://chaptersinwebsecurity.blogspot.com/2008/11/multiinjector-v03-released.html>

Mutz, D., Valeur, F., Vigna, G., & Kruegel, C. (2006). Anomalous system call detection. *ACM Transactions on Information Systems and Security, 9*, 61–93.

Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics, 33*, 1065–1076.

Patcha, A., & Park, J. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks, 51*(12), 3448–3470. URL: <http://www.sciencedirect.com/science/article/B6VRG-4N2KTNR-1/2/b8cc78a7f2e2f18946f4dfc2a66969ef>.

Puerto rico sites redirected in DNS attack security (April 2009). URL: <http://news.cnet.com/8301-1009_3-10228436-83.html>.

Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. *SIGMOD Record, 29*, 427–438.

Song, L., Smola, A., Gretton, A., Borgwardt, K. M., & Bedo, J. (2007). Supervised feature selection via dependence estimation. In *Proceedings of the 24th international conference on machine learning* (pp. 823–830). Corvalis, Oregon: ACM. doi:10.1145/1273496.1273600. URL: <http://portal.acm.org/citation.cfm?id=1273600>.

Tsai, C., Hsu, Y., Lin, C., & Lin, W. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications, 36*(10), 11994–12000. doi:10.1016/j.eswa.2009.05.029. URL <http://www.sciencedirect.com/science/article/B6V03-4WBC1NH-C/2/3e0271b6e5009bc945abd584bdb46c71>.

Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In: *Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles* (pp. 117–132). Big Sky, Montana, USA: ACM. doi:10.1145/1629575.1629587. URL: <http://portal.acm.org/citation.cfm?id=1629575.1629587&coll=portal&dl=ACM&type=series&idx=SERIES372&part=series&WantType=Proceedings&title=SOSP&CFID=59047162&CFTOKEN=30012674>.

Ye, N., Chen, Q., Emran, S. M., & Vilbert, S. (2000). Hotelling t2 multivariate profiling for anomaly detection. In *Proceedings of the 1st IEEE SMC information assurance and security workshop*.

Ye, N., Emran, S., Chen, Q., & Vilbert, S. (2002). Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on Computers, 51*(7), 810–820. doi:10.1109/TC.2002.1017701.

Ye, N., Li, X., Chen, Q., Emran, S., & Xu, M. (2001). Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 31*, 266–274.

Yeung, D. -Y., & Chow, C. (2002). Parzen-window network intrusion detectors. In: Proceedings of the 16th international conference on pattern recognition (Vol. 4, pp. 385–388).