McPAD : A Multiple Classifier System for Accurate Payload-based Anomaly Detection

Roberto Perdisci [a,b], Davide Ariu [c], Prahlad Fogla [d], Giorgio Giacinto [c], and Wenke Lee [b]

[a] *Damballa, Inc., Atlanta, 30308 GA, USA*

[b] *College of Computing, Georgia Institute of Technology, Atlanta, 30308 GA, USA*

[c] *University of Cagliari, 09123 Cagliari, Italy*

[d] *Google, Inc. Mountain View, CA 94043, USA*

Abstract

Anomaly-based network Intrusion Detection Systems (IDS) are valuable tools for the defense-in-depth of computer networks. *Unsupervised* or *unlabeled* learning approaches for network anomaly detection have been recently proposed. Such anomaly-based network IDS are able to detect (unknown) *zero-day* attacks, although much care has to be dedicated to controlling the amount of false positives generated by the detection system. As a matter of fact, it is has been shown [2] that the false positive rate is the true limiting factor for the performance of IDS, and that in order to substantially increase the Bayesian detection rate, $P(Intrusion|Alarm)$, the IDS must have a very low false positive rate (e.g., as low as 10^{-5} or even lower).

In this paper we present McPAD (Multiple-Classifier Payload-based Anomaly Detector), a new accurate payload-based anomaly detection system that consists of an ensemble of one-class classifiers. We show that our anomaly detector is very accurate in detecting network attacks that bear some form of shell-code in the malicious payload. This holds true even in the case of polymorphic attacks and for very low false positive rates. Furthermore, we experiment with advanced *polymorphic blending attacks* and we show that in some cases even in the presence of such sophisticated attacks and for a low false positive rate our IDS still has a relatively high detection rate.

Key words: Network Intrusion Detection, Anomaly Detection, Shell-Code Attacks, Multiple Classifiers, One-Class SVM

1 Introduction

Intrusion Detection Systems (IDS) are valuable tools for the defense-in-depth of computer networks. Network IDS look for known or potential malicious activities in network traffic and raise an alarm whenever a suspicious activity is detected. Two main approaches to intrusion detection are used, namely *misuse* and *anomaly* detection [25]. Misuse detectors are based on a description of known malicious activities. This description is often modeled as a set of rules referred to as *attack signatures*. Activities that match an attack signature are classified as malicious. On the other hand, anomaly detectors are based on a description of *normal* or *benign* activities. As malicious activities are expected to be different from normal activities, a suitable distance measure allows anomaly-based IDS to detect attack traffic.

The IDS most commonly used in real networks are signature-based, because they are able to efficiently detect known attacks while generating a relatively low number of false positives. Anomaly-based detection systems usually produce a relatively higher number of false positives, compared to the misuse-based or *signature-based* detection systems, because only a fraction of the anomalous traffic actually derives from intrusion attempts. Nevertheless, anomaly detectors are able to detect *zero-day* (i.e., never-before-seen) attacks, whereas signature-based systems are not.

Because it is very difficult and expensive to obtain a labeled dataset that is representative of real network activities and contains both normal and attack traffic [24], *unsupervised* or *unlabeled* learning approaches for network anomaly detection have been recently proposed [27, 12]. These methods aim to work on datasets of traffic extracted from real networks without the necessity of a labeling process. *Unlabeled* anomaly detection systems are based on the reasonable assumption that the percentage of attack patterns in the extracted traffic traces is usually much lower than the percentage of normal patterns [27]. Furthermore, it is possible to use signature-based IDS in order to filter the extracted traffic by removing the known attacks, thus further reducing the number of attack patterns possibly present in the dataset. Another assumption is that the attack patterns are supposed to be distinguishable from the normal patterns in a suitable feature space. The term “unlabeled anomaly detection” used in the intrusion detection field actually refers to what in machine learning is more often called “novelty detection”, “outlier detection” or “one-class classification”. One-class classification algorithms pursue concept learning in absence of counter examples [31], and have been shown to be promising for network anomaly detection [12].

1.1 Payload-based Anomaly Detection

Recent work on unlabeled anomaly detection focused on *high speed* classification based on simple *payload* statistics [19, 23, 35, 36] (the payload is the data portion of a network packet). For example, PAYL [35, 36] extracts 256 features from the payload. Each feature represents the occurrence frequency in the payload of one of the 256 possible byte values. A simple model of normal traffic is then constructed by computing the average and standard deviation of each feature. A payload is considered anomalous if a *simplified Mahalanobis distance* between the payload under test and the model of normal traffic exceeds a predetermined threshold. Although PAYL is based on simple statistics extracted from the payload, it has been shown to be quite effective [35]. Nonetheless, we show in Section 5 that PAYL may suffer from a relatively high false positive rate. Wang et al. [35] also proposed a more generic n-gram[1] version of PAYL. A sliding window of length n is used to extract the occurrence frequency in the payload of all the possible n-grams. In this case the payload is described by a pattern vector in a 256^n-dimensional feature space. The n-grams extract byte sequence information, which helps in constructing a more precise model of the normal traffic compared to the simple byte frequency-based model. The extraction of n-gram statistics from the payload can be performed efficiently and the IDS can be used to monitor high speed links in real time. However, given the exponentially growing number of extracted features, the higher n the more difficult it may be to construct an accurate model because of the curse of dimensionality and possible computational complexity problems.

Wang et al. also proposed ANAGRAM [37], an anomaly detector based on n-gram analysis that uses a different approach for modeling the traffic, compared to PAYL. ANAGRAM stores the distinct n-grams extracted from normal packets in a Bloom filter b_1, and the n-grams extracted from known attacks in a second Bloom filter b_2. During the test phase, for each packet all the distinct n-grams are extracted from the payload and compared with the filters b_1 and b_2. Payloads that contain too many n-grams that are not present in b_1 or that are present in b_2 are classified as anomalous [37]. Other anomaly detection systems based on more complex features have been proposed [33, 5]. These anomaly detectors involve the extraction of syntax and semantic information from the payload, which is usually a computationally expensive task. Therefore, it may not be possible to use this approach in order to analyze network traffic on high speed links in real time.

[1] Here an n-gram represents n consecutive bytes in the payload

1.2 Our Contribution: McPAD

In [2], Axelsson has shown that the false positive rate is the true limiting factor for the performance of IDS, and that in order to substantially increase the Bayesian detection rate, $P(Intrusion|Alarm)$ (i.e., the probability of having an intrusion given that an alarm was raised), the false positive rate of the IDS must be very low (e.g., as low as 10^{-5} or even lower). Although fairly effective, payload-based anomaly detectors like PAYL [35] suffer from a relatively high false positive rate, and may therefore have a very low Bayesian detection rate.

Our goal is to devise a new payload-based anomaly detector that uses simple payload statistics to accurately detect network attacks, and in particular shell-code attacks [1] (i.e., attacks that inject executable code), even at a very low false positive rate. We address these challenge using an ensemble of classifiers. Classifier ensembles, often referred to as Multiple Classifier Systems (MCS), have been proved to achieve a better trade-off between false positive and detection rate in many applications, compared to the best single classifier in the ensemble [10, 20]. A number of security related applications of MCS have been proposed in the literature. For example, MCS are used in multimodal biometrics for hardening person identification [4], and in misuse-based IDS [16] to improve the detection accuracy.

MCS attain accuracy improvements when the combined classifiers are “diverse”, i.e., they make different (ideally independent) errors on new patterns [10]. A way to induce diversity is to combine classifiers that are based on descriptions of the patterns in different feature spaces [20]. The combination of classifiers trained on different feature spaces allows us to effectively exploit the complementarities of the different pattern representations [10]. In [26] we proposed a new approach to construct a *high speed* payload-based anomaly IDS by combining multiple one-class Support Vector Machine (SVM) classifiers using a majority voting rule. As mentioned in Section 1.1, n-gram analysis has been shown to be quite effective for payload-based anomaly detection. However, the exponential increase of the dimensionality of the features space may make it hard to accurately model the normal traffic for values of $n > 2$. In order to solve this problem, we proposed a new technique to extract the features from the payload that is similar to the 2-gram technique [26]. Instead of measuring the frequency of the pairs of consecutive bytes, we proposed to measure the features by using a sliding window that “covers” two bytes which are ν positions apart from each other in the payload. We refer to these pairs of bytes as 2_ν-grams. The proposed feature extraction process does not add any complexity with respect to the traditional 2-gram technique and can be performed efficiently. We also showed that the proposed technique allows us to “summarize” the occurrence frequency of n-grams, with $n > 2$, thus capturing byte sequence information while limiting the dimensionality of the feature space. By varying the parameter ν, we constructed a representation of the payload in different feature spaces. Also, we adapted a feature clustering algorithm originally proposed in [9] to one-class problems, and used it to reduce the dimensionality of the different feature spaces where the payload is represented. We obtained high detection accuracy at very low false positive rate by constructing our anomaly-based IDS, which we called McPAD (Multiple-classifier Payload-based Anomaly Detector), using a combination of multiple one-class SVM classifiers that work on these different feature spaces.

In this paper we extend the results reported in [26]. We present a much deeper analysis of our anomaly detection system and its capabilities regarding detection accuracy for shell-code attacks [1] and polymorphic shell-code attacks [30]. Shell-code attacks are among the most dangerous kinds of network attacks because they carry executable malicious code that can hijack the normal execution of the target vulnerable application. When successful, shell-code attacks often allow the attacker to gain complete control of the victim machine. Polymorphic shell-code attacks automatically create new variants of the attack code before sending the actual intrusion attempt to a new victim, and are therefore very difficult to detect using signature-based IDS [30].

We experiment with different classifier combination schemes on large datasets of both normal traffic and attacks. In particular, we perform experiments on the first week of traffic from the DARPA'99 dataset [22], and on seven days of real HTTP traffic collected at an academic institution. As attack dataset, we use a large dataset of "standard" HTTP attacks provided by the authors of [17], which is publicly available. Furthermore, we construct a large number of polymorphic attacks, which include attacks generated with the polymorphic shell-code engine CLET [8], a set of Polymorphic Blending Attacks (PBA) [14, 13] designed to evade PAYL [35], and a set of PBA specifically designed with the intent to evade our detection system. We compare McPAD to PAYL [35], and we show that our IDS has a much higher detection accuracy than PAYL on shell-code attacks at low false positive rates, and is in some cases resistant to even the sophisticated polymorphic blending attacks specifically designed to evade it. Furthermore, we released the source code of McPAD and the attack datasets we used for our experiments, with the hope of making the results we obtained reproducible.

The remainder of the paper is organized as follows. In Section 2 we summarize the most relevant related work, whereas in Section 3 we give some background on one-class classification and the combination of one-class classifier. In Section 4 we present the details of our McPAD detection system, and in Section 5 we present and discuss our experimental results. We then briefly conclude in Section 6.

2 Related Work

A number of payload-based anomaly IDS have been proposed which monitor the payload of a packet for anomalies. In [19] Kruegel et al. describe a service-specific intrusion detection system. They combine the type, length and payload byte distribution of the service requests as features in a statistical model of normal traffic to compute an anomaly score.

NETAD [23] monitors the first 48 bytes of IP packets. A number of separate models are constructed corresponding to the most common network protocols. An anomaly score is computed in order to detect rare events.

As mentioned before, PAYL [35] measures the frequency of the n-grams in the payload (i.e., the data portion of the packet) and constructs one model of normal traffic for each different packet length. A new version of PAYL presented in [36] adds some functionalities to the original version. In particular, the new version constructs a number of models for each packet length and performs inbound and outbound traffic correlation to detect the propagation of worms.

Polymorphism is commonly used in computer virus toolkits and is becoming more and more used by worm writers. *Evasive polymorphism* has been recently explored [8, 14]. CLET [8], an advanced polymorphic engine, performs spectrum analysis to evade IDS that use data mining methods for worm detection. Given an attack payload, CLET adds padding bytes in a separate *cramming bytes* zone (of given length) to make the byte frequency distribution of the attack close to the model of normal traffic. In [14], Fogla et al. proposed a *polymorphic blending attack*, which uses byte substitution and padding techniques to evade PAYL. In [13] Fogla and Lee present a formal analysis of the polymorphic blending attack, and in [30] Song et al. discuss the infeasibility of modeling polymorphic shell-code and concludes that signature-based approaches cannot be successful in case of sophisticated polymorphic attacks.

In order to cope with polymorphic blending attacks, Wang et al. proposed a new payload-based anomaly detection system called ANAGRAM [37]. ANAGRAM stores n-grams that appear in normal traffic into a Bloom filter. During detection, the n-grams are extracted from the payload under test. If more than a certain percentage of n-grams are not present in the Bloom filter constructed during training, the payload is classified as anomalous. Although similar in the motivation, our approach is different from ANAGRAM because we model only normal traffic, and we use multiple one-class SVM classifiers [28] to construct our anomaly detector.

One of the first works on unlabeled network anomaly detection was presented in [27], where a clustering algorithm is used to discover outliers in the training dataset. Once the normal patterns are separated from outlier patterns, the clusters of normal data are used to construct a supervised detection model. In [12], Eskin et al. presented a geometric framework to perform anomaly detection. The patterns are mapped from a feature space F to a new feature space F' and anomalies are detected by searching for patterns that lie in sparse regions of F'. Three different classification technique are used, a clustering algorithm, a k-NN algorithm and a SVM-based one-class classifier. Experiments are performed on the KDD-UCI dataset, both on the portion containing network traffic, and on the portion containing sequences of system-calls.

Recently the paradigm of Multiple Classifier Systems (MCS) has been proposed for misuse detection in [16] where classifiers trained on different feature subsets are combined to attain better performance than those attained by classifiers trained on the large feature space made of the union of subsets. A different approach is proposed in [6] where a serial combination of classifiers is proposed. Network traffic is serially processed by different classifiers. At each stage a classifier may either decide for one attack class or send the pattern to another stage, which is trained on more difficult cases. Reported results show that MCS improve the performance of IDS based on statistical pattern recognition techniques.

Our work was mainly inspired by [35] and [14]. We explore MCS constructed by combining one-class SVM classifiers [28] for unlabeled payload-based anomaly detection. We propose a new technique derived from n-gram analysis to extract statistical features from the payload. The combination of multiple classifiers aims at improving the classification performance of payload-based anomaly IDS and to make polymorphic attacks harder to succeed.

3 Background

McPAD, our anomaly detector, is based on an ensemble of one-class Support Vector Machine (SVM) classifiers. We will present the details of McPAD in Section 4. In this section we provide some background on one-class classification that is necessary for an easier understanding of the rest of the paper. We first discuss one-class classification in general, then we present the internals of one-class SVM classifiers, and the approach we adopt to combine them using different combination rules.

3.1 One-Class Classification

One-class classification techniques are particularly useful in case of *two-class* learning problems whereby one of the classes, referred to as *target class*, is well-sampled, whereas the other one, referred to as *outlier class*, is severely undersampled. The low number of examples from the outlier class may be motivated by the fact that it is too difficult or expensive to obtain a significant number of training patterns of that class [31]. The goal of one-class classification is to construct a decision surface around the examples from the target class in order to distinguish between *target objects* and all the other possible objects, i.e., the *outliers* [31]. Given an unlabeled training dataset that is deemed to contain mostly target objects, a *rejection rate* is usually chosen during training so that a certain percentage of training patterns lies outside the constructed decision surface. This takes into account the possible presence of noise (i.e., unlabeled outliers), and allows us to obtain a more precise description of the target class [31]. In the case when the training set contains only "pure" target patterns, this rejection rate can be interpreted as a *tolerable* false positive rate.

Several different one-class classification algorithms have been proposed in the literature [31]. In the following, we briefly describe the One-Class SVM presented by Schölkopf et al. in [28], which we use to create the ensemble of classifiers that constitutes the core of McPAD. As we discuss in Section 4, there is an analogy between anomaly detection based on n-gram statistics and text classification problems. We chose the one-class SVM classifier because SVM have been shown to achieve good performance in text classification problems [29, 21]. Afterwards we explain how we combine One-Class SVM classifiers using combination rules such as probability average, product, maximum, and minimum, and the majority voting algorithm.

We will refer to a pattern vector $\mathbf{x}_k = [x_{k_1}, x_{k_2}, .., x_{k_l}]$ as the description of a payload π_k (i.e., the data portion of a network packet) in a l-dimensional feature space $\mathbb{F}$. We discuss in Section 4 how the features $x_{k_i},\ i = 1, .., l$, are measured in practice.

3.2 One-Class SVM

A one-class classifier inspired by the Support Vector Machine (SVM) classifier [34] was proposed by Schölkopf et al. in [28]. The one-class classification problem is formulated to find a hyperplane that separates a desired fraction of the training patterns, called the *target* patterns, from the origin of the feature space $\mathbb{F}$. This hyperplane cannot be always found in the original feature space, thus a mapping function $\Phi : \mathbb{F} \rightarrow \mathbb{F}'$, from $\mathbb{F}$ to a kernel space $\mathbb{F}'$, is used. In particular, it can be proven that when the gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) = exp\left(-\gamma ||\mathbf{x} - \mathbf{y}||^2\right) \tag{1}$$

is used it is always possible to find a hyperplane that solves the separation problem. The problem is formulated as follows:

$$\begin{gathered} \min_{\mathbf{w},\xi,\rho}\left(\tfrac{1}{2}\left\|\mathbf{w}\right\|^2 - \rho + \tfrac{1}{hC}\sum_i \xi_i\right) \\ \mathbf{w} \cdot \phi\left(\mathbf{x}_i\right) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, .., h \end{gathered} \tag{2}$$

where $\mathbf{w}$ is a vector orthogonal to the separation hyperplane, C represents the fraction of training patterns that are allowed to be rejected (i.e., that are not separated from the origin by the hyperplain), $\mathbf{x}_i$ is the i-th training pattern, h is the total number of training patterns, $\xi = [\xi_1, .., \xi_h]$ is a vector of slack variables used to "penalize" the rejected patterns, ρ represents the margin, i.e., the distance of the hyperplane from the origin.

The solution of (2) gives us the desired separation hyperplane. A generic test pattern $\mathbf{z}$ can then be classified as *target* or *outlier* using the following decision function [28]

$$f_{svc}(\mathbf{z}) = I\left(\sum_i \alpha_i K\left(\mathbf{x}_i, \mathbf{z}\right) \geq \rho\right), \quad \textstyle\sum_{i=1}^{h} \alpha_i = 1 \tag{3}$$

where I is the indicator function (whereby $I(x) = 1$ if x is true, otherwise $I(x) = 0$), and the coefficients α_i and the threshold ρ are provided by the solution of (2). According to (3), a pattern $\mathbf{z}$ is either rejected (i.e., classified as *outlier*) if $f_{svc}(\mathbf{z}) = 0$, or accepted as *target* object if $f_{svc}(\mathbf{z}) = 1$. It is worth noting that most of the coefficients α_i are usually equal to zero, therefore $f_{svc}(\mathbf{z})$ can be efficiently computed. The training patterns $\mathbf{x}_i$ for which $\alpha_i \neq 0$ are referred to as *support vectors.*

3.3 Combining Multiple One-Class SVM Classifiers

Unlike the combination of two-class or multi-class classifiers, the combination of one-class classifiers is usually not straightforward [32]. This is due to the fact that usually it is not possible to reliably estimate the probability distribution of the outlier class. As a consequence the posterior class probability cannot be estimated and many combination rules used in multi-class classification problems may not be applied. However, in [15] we showed that, when the Gaussian kernel (1) is used, the output of the one-class SVM can be formulated in terms of a class conditional probability by

$$p(\mathbf{x}|\omega_t) = \frac{1}{(2\pi \cdot s)^{\frac{d}{2}}} \sum_{i=1}^{h} \alpha_i \cdot K(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{h} \alpha_i \frac{1}{(2\pi \cdot s)^{\frac{d}{2}}} \cdot e^{-\frac{1}{2}\frac{||\mathbf{x}-\mathbf{x}_i||^2}{s}} \quad (4)$$

which respects the constraint $\int_{\mathbb{R}^d} p(\mathbf{x}|\omega_t)d\mathbf{x} = 1$ [15]. Assuming a uniform distribution for the outlier class, this allows us to combine L different one-class SVM classifiers as

$$y_{avg}(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L} p_i(\mathbf{x}|\omega_t) \quad (5)$$

for example, where ω_t represents the target class. We can then use the simple decision criterion [15]

$$y_{avg}(\mathbf{x}) < \theta \;\Rightarrow\; \mathbf{x} \text{ is an outlier} \quad (6)$$

where θ is a predefined threshold that can be tuned to find the desired trade-off between false positives and detection rate. Equations (5) and (6) represent the *average of probabilities* combination rules. Other alternative combination rules are

$$y_{prod}(\mathbf{x}) = \prod_{i=1}^{L} p_i(\mathbf{x}|\omega_t), \quad y_{prod}(\mathbf{x}) < \theta \;\Rightarrow\; \mathbf{x} \text{ is an outlier} \quad (7)$$

which is the *product of probabilities* rule, and

$$y_{min}(\mathbf{x}) = \min_{i=1}^{L} p_i(\mathbf{x}|\omega_t), \quad y_{min}(\mathbf{x}) < \theta \;\Rightarrow\; \mathbf{x} \text{ is an outlier} \quad (8)$$

and

$$y_{max}(\mathbf{x}) = \max_{i=1}^{L} p_i(\mathbf{x}|\omega_t), \quad y_{max}(\mathbf{x}) < \theta \;\Rightarrow\; \mathbf{x} \text{ is an outlier} \quad (9)$$

which are the *minimum* and *maximum probability* combination rules, respectively. Average, product, maximum and minimum probabilities are popular simple (low cost) non-trainable combiners that have been show to be quite successful for different classification problems [18, 20]. However, it is often difficult to predict which classification rule will perform the best on a specific real problem. This is the reason why in the following we will compare the results obtained using different rules for combining the output of different classifiers in our anomaly detection system.

It is worth noting that in general only a small number of coefficients α_i will be different from zero, thus $p(\mathbf{x}|\omega_t)$ can be efficiently computed.

In case of the majority voting rule there is no need to estimate the class conditional probabilities, and the application of the combination rule is straightforward. Assume the output of the L classifiers related to a payload π_k to be given as a vector $c(\pi_k) = [c_1(\mathbf{x}_k^{(1)}), c_2(\mathbf{x}_k^{(2)}), .., c_L(\mathbf{x}_k^{(L)})] \in \{0,1\}^L$, where

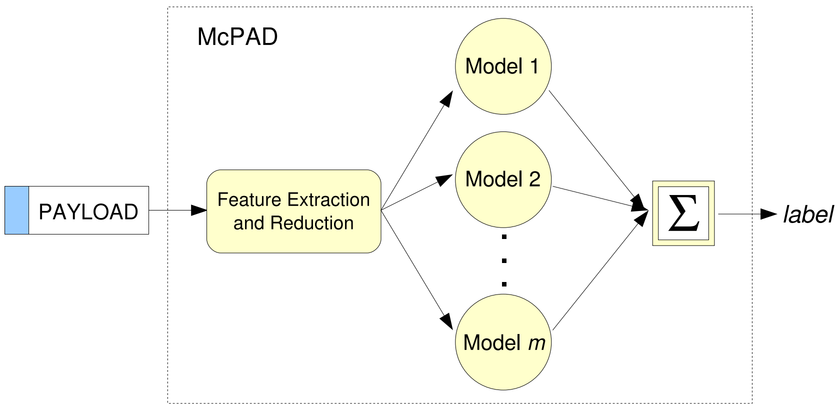

Fig. 1. Overview of McPAD

$c_h(\mathbf{x}_k^{(h)}) = 1$ if the h-th classifiers labels π_k as target, otherwise $c_h(\mathbf{x}_k^{(h)}) = 0$. The majority voting rule can be written as $\sum_{i=1..L} c_i(\mathbf{x}_k^{(1)}) > L/2$.

4 McPAD

In this section we provide the details of McPAD. We first describe how McPAD extracts the features, and the algorithm used for dimensionality reduction. Afterwards, we describe how different models of normal traffic are combined to build our multiple classifier anomaly detector, and we formally analyze the complexity of our classification system. A simplified view of McPAD is depicted in Figure 1.

4.1 Feature Extraction

As we mentioned above, the detection model used by PAYL [35] is based on the frequency distribution of the n-grams (i.e., the sequences of n consecutive bytes) in the payload. The occurrence frequency of the n-grams is measured by using a sliding window of length n. The window slides over the payload with a step equal to one byte and counts the occurrence frequency in the payload of the 256^n possible n-grams. Therefore, in this case the payload is represented by a pattern vector in a 256^n-dimensional feature space. It is easy to see that the higher n, the larger the amount of structural infomation extracted from the payload. However, using $n = 2$ we already obtain 65,536 features. Larger values of n are impractical given the exponentially growing dimensionality of the feature space and the curse of dimensionality problem [11].

In order to solve this problem we propose to measure the occurrence frequency of pairs of bytes that are ν positions (i.e., ν bytes) apart from each other in the payload. This allows us to efficiently extract some information related to the n-grams, with $n > 2$. We call such pairs of bytes 2_ν*-grams*. Regardless of the value of the parameter ν, measuring the 2_ν-gram extracts 256^2 features. As we will discuss in the following, by combining information extracted by using 2_ν-grams with different values of ν we can somehow (partially) reconstruct the information that we would extract by directly measuring the frequency of n-grams, with $n > 2$.

In practice, the occurrence frequency of the 2_ν-grams can be measured by using a $(\nu + 2)$ long sliding window with a "gap" between the first and last byte. Consider a payload $B = [b_1, b_2, .., b_l]$, where b_i is the byte value at position i. The occurrence frequency in the payload B of an n-gram $\beta = [\beta_1, \beta_2, .., \beta_n]$, with $n < l$, is computed as

$$f(\beta|B) = \frac{\#\text{ of occurrences of }\beta\text{ in }B}{l - n + 1} \tag{10}$$

where the number of occurrences of β in B is measured by using the sliding window technique, and $(l - n + 1)$ is the total number of times the window can "slide" over B. $f(\beta|B)$ can be interpreted as an estimate of the probability $p(\beta|B)$ of finding the n-gram β (i.e., the sequence of consecutive bytes $[\beta_1, \beta_2, .., \beta_n]$) in B. Accordingly, the probability of finding a 2_ν-gram $\{\beta_1, \beta_{\nu+2}\}$ can be written as

$$p(\{\beta_1, \beta_{\nu+2}\}|B) = \sum_{\beta_2,..,\beta_{\nu+1}} p([\beta_1, \beta_2, .., \beta_{\nu+1}, \beta_{\nu+2}]|B) \tag{11}$$

where the summation is over all the possible combinations of $\beta_2, .., \beta_{\nu+1}$. It is worth noting that for $\nu = 0$ the 2_ν-gram technique reduces to the "standard" 2-gram technique. When $\nu > 0$, the occurrence frequency in the payload of a 2_ν-gram $\{\beta_1, \beta_{\nu+2}\}$ can be viewed as a marginal probability computed on the distribution of the $(\nu + 2)$-grams that start with β_1 and end with $\beta_{\nu+2}$.

From the occurrence frequency of the n-grams it is possible to derive the distribution of the $(n-1)$-grams, $(n-2)$-grams, etc. On the other hand, measuring the occurrence frequency of the 2_ν-grams does not allow us to automatically derive the distribution of $2_{(\nu-1)}$-grams, $2_{(\nu-2)}$-grams, etc. The distributions of 2_ν-grams with different values of ν give us different structural information about the payload. The intuition is that, ideally, if we could somehow combine the structural information extracted using different values of $\nu = 0, .., N$ we would be able to (at least partially) reconstruct the structural information given by the distribution of n-grams, with $n = (N + 2)$. This intuition motivates the combination of classifiers that work on different descriptions of the payload obtained using the 2_ν-gram technique with different values of ν.

4.2 Dimensionality Reduction

Payload anomaly detection based on the frequency of n-grams is analogous to a text classification problem for which the bag-of-words model and a simple unweighted raw frequency vector representation [21] is used. The different possible n-grams can be viewed as the words, whereas a payload can be viewed as a document to be classified. In general for text classification only the words that are present in the documents of the training set are considered. This approach is not suitable in case of a one-class classification problem. Given that the training set contains (almost) only target examples (i.e., "normal" documents), we cannot conclude that a word that has a probability equal to zero to appear in the training dataset will not be discriminant. As a matter of fact, if we knew of a word w that has probability $p(w|d_t) = 0,\ \forall d_t \in C_t$, of appearing in the class of target documents C_t, and $p(w|d_o) = 1,\ \forall d_o \in C_o$, of appearing in documents of the outlier class C_o, it would be sufficient to measure just one binary feature, namely the presence or not of w_t in the document, to construct a perfect classifier. This is the reason why we choose to take into account all the 256^n n-grams, even though their occurrence frequency measured on the training set is equal to zero. Using the 2_ν-gram technique we still extract 256^2 features. This high number of features could make it difficult to construct an accurate classifier, because of the curse of dimensionality [11] and possible computational complexity problems related to learning algorithms.

In order to reduce the dimensionality of the feature space for payload anomaly detection, we apply a feature clustering algorithm originally proposed by Dhillon et al. in [9] for text classification. Given the number of desired clusters k, which is chosen a priori, the algorithm first randomly splits the features into k groups. Then, the features are iteratively moved from one of the k clusters to another until the information loss due to the clustering process is less than a certain threshold τ. This clustering algorithm has the property to reduce the within cluster and among clusters Jensen-Shannon divergence [9] computed on the distribution of words, and has been shown to help obtain better classification accuracy results with respect to other feature reduction techniques for text classification [9]. The inputs to the algorithm are:

(1) The set of distributions $\{p(C_i|w_j) : 1 \le i \le m,\ 1 \le j \le l\}$, where C_i is the i-th class of documents, m is the total number of classes, w_j is a word and l is the total number of possible different words in the documents.
(2) The set of all the priors $\{p(w_j),\ 1 \le j \le l\}$.
(3) The number of desired clusters k.
(4) The tolerable information loss τ.

The output is represented by the set of word clusters $W = \{W_1, W_2, .., W_k\}$. Therefore, after clustering the dimensionality of the feature space is reduced from l to k. In the original l-dimensional feature space, the j-th feature of a pattern vector $\mathbf{x}_i$ represents the occurrence frequency $f(w_j|d_i)$ of the word w_j in the document d_i. The new representation $\mathbf{x}'_i$ of d_i in the k-dimensional feature space can be obtained by computing the features according to

$$f(W_h|d_i) = \sum_{w_j \in W_h} f(w_j|d_i), \quad h = 1,..,k \tag{12}$$

where $f(W_h|d_i)$ can be interpreted as the occurrence frequency of the cluster of words W_h in the document d_i.

In case of a one-class problem, $m = 2$ and we can call C_t the target class and C_o the outlier class. The posterior probabilities $\{p(C_i|w_j) : i = t, o,\ 1 \leq j \leq l\}$ can be computed as

$$p(C_i|w_j) = \frac{p(w_j|C_i)p(C_i)}{p(w_j|C_t)p(C_t)+p(w_j|C_o)p(C_o)} \qquad i = t, o, \quad 1 \leq j \leq l \tag{13}$$

and the priors $\{p(w_j),\ 1 \leq j \leq l\}$ can be computed as

$$p(w_j) = p(w_j|C_t)p(C_t) + p(w_j|C_o)p(C_o), \quad 1 \leq j \leq l \tag{14}$$

The probabilities $p(w_j|C_t)$ of finding a word w_j in documents of the target class C_t can be reliably estimated on the training dataset, whereas it is difficult to estimate $p(w_j|C_o)$, given the low number (or the absence) of examples of documents in the outlier class C_o. Similarly, it is difficult to reliably estimate the prior probabilities $p(C_i) = \frac{N_i}{N},\ i = t, o$, where N_i is the number of training patterns of the class C_i and $N = N_t + N_o$ is the total number of training patterns. Given that $N_o \ll N_t$ (or even $N_o = 0$), the estimated priors are $p(C_o) \simeq 0$ and $p(C_t) \simeq 1$, which may be very different from the real prior probabilities.

In our application, the words w_j are represented by the 256^2 possible different 2_ν-grams (with a fixed ν). In order to apply the feature clustering algorithm, we estimate $p(w_j|C_t)$ by measuring the occurrence frequency of the 2_ν-grams w_j on the training dataset and we assume a uniform distribution $p(w_j|C_o) = \frac{1}{l}$ of the 2_ν-grams for the outlier class. We also assume $p(C_o)$ to be equal to the desired rejection rate for the one-class classifiers (see Section 3), and accordingly $p(C_t) = 1 - p(C_o)$.

4.3 Payload Classification

By varying the parameter ν and applying the dimensionality reduction algorithm explained above, we obtain different compact representations of the payload in different feature spaces. For each of these representations we construct a model of normal traffic by training a one-class SVM classifier (see Section 3.2). In practice, given a set of "gap" values $\{\nu_i\}_{i=1..m}$, and a dataset $D = \{p_k\}_{k=1..N}$ of (mostly) normal traffic, we construct m datasets $D^{\nu_i} = \{p_k^{\nu_i}\}_{k=1..N}, i = 1..m$, one for each ν_i, according to the feature extraction and dimensionality reduction process described in Section 4.1 and Section 4.2. Afterwards, we train a one-class SVM classifier on each dataset D^{ν_i}, thus obtaining m models $M_1, M_2, .., M_m$ of normal traffic.

During the operational phase, whenever a test payload p is received, we compute m different representations of p, namely $p^{\nu_1}, p^{\nu_2}, ..p^{\nu_m}$ according to the feature extraction and dimensionality reduction process described above. Then, we classify each representation p^{ν_i} using model M_i. Finally, we combine the classification results obtained from each model in order to make a final decision. We use the combination approach described in Section 3.3. In the case of the average, product, minimum and maximum probability, the output of the combiner is a score that can be interpreted as the probability of the payload under test being normal, $P(normal|p)$ (see Section 3.3). The final decision depends on a threshold θ, whereby the payload p is classified as an *attack* if $P(normal|p) < \theta$, and as *normal* otherwise. In case of the majority voting combination rule (see Section 3.3), the output of the combiner equals the number of classifiers that labeled p as *attack*. In this case, we can again set a threshold θ, whereby if the number of classifiers that deemed p as *attack* is greater than θ the final decision will be to classify p as *attack*, otherwise it will be classified as *normal*. The threshold θ can be chosen in order to tune the inevitable trade-off between false positives and detection rate.

4.4 Complexity Analysis

In this section we provide an analysis of the computational complexity of our McPAD detection algorithm. Because the training of McPAD can be performed off-line, here we only present an analysis of the computation involved in the test phase.

Given a payload π_j of length n and a fixed value of ν, the frequency of 2_ν-grams (see Section 4.1) can be computed in $O(n)$. As the number of extracted features is constant (equal to 2^{16}, regardless of the actual value of n and ν), the mapping between the frequency distribution of 2_ν-grams and the k feature clusters (see Section 4.2) can be computed using a simple look-up table and a number of sum operations that is always less than 2^{16} (regardless of the value of k). Therefore the feature reduction process can be computed in in $O(1)$. The feature extraction and reduction process has to be repeated m times choosing every time a different value of ν. m represents the number of different one-class classifiers used to make a decision about each payload π_j, and the overall feature extraction and reduction process can be accomplished in $O(nm)$.

Once the features have been extracted, and the dimensionality reduced to k, each payload has to be classified according to each of the m one-class SVM classifiers. Let us first consider one single classifier. As mentioned in Section 3.2, in order to classify a pattern vector x_j (i.e., a representation of the payload π_j, in our case), the distance between x_j and each of the support vectors obtained during training has to be computed. Given the number of feature clusters k, and the number of support vector s, the classification of a pattern can be computed in $O(ks)$. This classification process has to be repeated m times and the results are then combined (e.g., using the average of probability). The overall classification of a pattern vector x_j can therefore be computed in $O(mks)$. It is worth noting that the number of support vectors s depends on two main parameters, namely the size t of the training set, and the value of the parameter C in Equation (2). The parameter C can be interpreted as the fraction of "desired" false positives. Schölkopf et al. [28] showed that when the solution of the problem in Equation (2) satisfies $\rho \neq 0$, $tC \leqslant s$. Therefore, for a fixed size t of the training dataset the higher the "desired" false positive rate chosen during training, the higher s, and in turn the higher the computation time spent on each packet. We will show in Section 5 that this is the source of an inevitable trade-off between the accuracy of McPAD and its average computational cost per payload.

5 Experiments

In this section we report the results of the extensive experiments we performed. We present the results regarding the accuracy of McPAD first, and then we compare it to PAYL [35]. We limited our analysis to HTTP traffic that consists of several days of simulated and real legitimate HTTP requests, several HTTP attacks provided by the authors of [17], and a large number of polymorphic HTTP attacks we generated ourselves. We found that collecting a sufficient amount of attack traffic for protocols other than HTTP is very hard and expensive. To the best of our knowledge no such dataset is publicly available.

Table 1
Summary of the parameter settings used in the experiments with McPAD.

γ	0.5
ν	0-10
Feature Clusters (k)	10, 20, 40, 80, 160
Desired FP Rate	10%, 5%, 2%, 1%, 0.5%, 0.2% 0.1%, 0.05%, 0.02%, 0.01%, 0.001%

5.1 Experimental Setup

We implemented an open-source proof-of-concept version of McPAD, which we made available at `http://roberto.perdisci.googlepages.com/mcpad`. McPAD is written entirely in Java and is built on top of LibSVM (`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`) and Jpcap (`http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`). In order to compare McPAD to PAYL [35], we requested and obtained a copy of the PAYL software from Columbia University.

We experimented with several different combinations of the configuration parameters for McPAD. We used a "gap" $\nu = 0..10$ for the 2_ν-gram feature extraction process (see Section 4.1). We used one-class SVM classifiers with Gaussian Kernel (see Section 3.2). The value of γ for the Gaussian Kernel was set equal to 0.5 in all the experiments, because this value gave good results during preliminary experiments [26]. We used several values for the number of feature clusters k used in the dimensionality reduction process (see Section 4.2). In particular we experimented with $k = 10, 20, 40, 80, 160$ feature clusters. The "desired" false positive rate (see Section 3) used during the training of the one-class SVM classifiers was set to $FP = 10\%, 5\%, 2\%, 1\%, 0.5\%, 0.2\%, 0.1\%, 0.05\%, 0.02\%, 0.01\%, 0.005\%$, 0.002%, and 0.001%.

The parameter values used during the experiments are summarized in Table 1, for convenience.

5.2 Validation Metrics

In order to validate the classification performance of our detector, we use the Receiver Operating Characteristic (ROC) curve analysis, and the Area Under the Curve (AUC). The ROC curve provides a way to visually represent how the trade-off between false positives and detection rate varies for different values of the detection threshold [3]. Differently from the classic accuracy metric, which suffers from a dependency to specific values of the detection threshold and related false positives and detection rate [3], the AUC summarizes the classification performance of the classifier in the entire range $[0, 1]$ of the false positive rate and can be interpreted as the probability of scoring attack packets higher than legitimate packets [7] (i.e., the higher the AUC, the easier to distinguish attacks from normal traffic).

One problem with the AUC for evaluating intrusion detection systems is that it is computed along the entire range $[0, 1]$ of the false positive rate. Because it is not realistic that an intrusion detection system will be configured to generate a high number of false alarms, we are mainly interested in evaluating the classification performance of our anomaly detector for low values of the false positive rate. To this end, we compute the area under the ROC curve in the range $[0, 0.1]$ of the false positive rate (i.e., we do not take into account how the classification system performs for a false positive rate higher than 10%). We normalize the "partial" AUC computed in $[0, 0.1]$ by dividing it by 0.1, in order to obtain a number that varies between 0 and 1. The highest the value of the normalized AUC the better the classification performance.

5.3 Datasets

In this section we describe the characteristics of the datasets that we used in our experiments.

DARPA We used the HTTP requests extracted from the first week of the DARPA'99 dataset [22], which consists of five entire days of simulated normal traffic to and from an airforce base. Although the DARPA dataset is outdated and has been criticized [24, 23] for the way it was generated, to the best of our knowledge it is the only public dataset of network traffic that represents a common base on which experimental results may be reproduced and compared to the ones obtained using different approaches. We randomly split each day of traffic into two parts, a training set made of approximately 80% of the traffic and a validation set made of the remaining 20% of the traffic. For each day of training/validation, a test dataset was constructed which consists of 20% of the traffic randomly chosen from all the remaining days (i.e., the traffic from the days which are not included in the training/validation set). The characteristics of the obtained dataset are reported in Table 2.

GATECH We collected seven days of real HTTP requests towards the website of the College of Computing School at the Georgia Institute of Technology.

Table 2
DARPA Dataset Characteristics

DARPA	**Training Set**		**Validation Set**		**Test Set**	
Day	**Size (MB)**	**Packets**	**Size (MB)**	**Packets**	**Size (MB)**	**Packets**
1	19	161,602	4.7	40,057	44	137,997
2	23	196,605	5.7	48,905	42	131,738
3	23	189,362	5.5	46,957	42	133,133
4	30	268,250	7.6	67,593	39	121,999
5	18	150,847	4.4	37,639	45	139,869

Table 3
GATECH dataset characteristics.

GATECH	**Training Set**		**Validation Set**		**Test Set**	
Day	**Size (MB)**	**Packets**	**Size (MB)**	**Packets**	**Size (MB)**	**Packets**
1	131	307,929	33	76,654	147	350,849
2	72	171,750	19	43,418	162	385,247
3	124	289,649	31	72,320	149	354,637
4	110	263,498	28	65,260	152	361,789
5	79	195,192	20	48,653	161	379,610
6	78	184,572	20	45,949	160	380,895
7	127	296,425	32	74,218	148	352,119

Although this traffic is completely unlabeled, it is very reasonable to consider it as containing mostly legitimate traffic. This would not be true only in case persistent intrusion attempts were ongoing at the time we collected the traffic. Such attacks are generally unlikely and usually noticeable. Given that no evidence of persistent attacks was reported during the period in which we collected the traffic, we speculate the level of noise in our dataset is negligible. Therefore, in the following we consider the GATECH dataset as "clean" for the purpose of measuring the false positive rate. Similarly to the DARPA dataset, we divided each day of traffic into two parts, a training set made of approximately 80% of the traffic and a validation set made of 20% of the traffic. For each day of training/validation, a test dataset was constructed which consists of 20% of the traffic from all the remaining days. The characteristics of the GATECH dataset are reported in Table 3.

ATTACKS We experimented with several non-polymorphic and polymorphic HTTP attacks. Although we were able to find a public source of non-polymorphic HTTP attacks provided by the authors of [17], we were not able to find any public source of polymorphic attacks. We therefore created the polymorphic attacks ourselves, using both the polymorphic engine CLET [8] and a Polymorphic Blending Attack engine similar to the one used in [14, 13]. We decided to make the entire attack dataset publicly available at `http://roberto.perdisci.googlepages.com/mcpad`, in the hope that this will foster future research. We divided the attack dataset into the following groups of attacks:

- **Generic Attacks**. This dataset includes all the HTTP attacks provided by the authors of [17] plus a shell-code attack that exploits a vulnerability (MS03-022) in Windows Media Service (WMS), which we used in [26]. In total this dataset consists of 66 HTTP attacks. Among these, 11 are shell-code attacks, i.e., attacks that carry executable code in the payload. Other attacks cause Information Leakage and Denial of Service (DoS), for example.
- **Shell-code Attacks**. This dataset contains 11 shell-code attacks from the *Generic Attacks* dataset. Shell-code attacks are particularly dangerous because their objective is to inject executable code and hijack the normal execution of the target application. Some famous worms, like Code-Red, for example, use shell-code attacks to propagate.
- **CLET Attacks**. This dataset contains 96 polymorphic attacks generated using the polymorphic engine CLET [8]. We selected 8 among the 11 *Shell-code Attacks* for which the source code was available, and created a polymorphic version of each attack using the payload statistics computed on each distinct day of traffic from the DARPA and GATECH datasets for training CLET's polymorphic engine. Overall we generated 96 polymorphic CLET attacks.
- **Polymorphic Blending Attacks (PBAs)**. We created this dataset using three shell-code attacks, namely Code-Red (a famous worm that exploits a vulnerability in Windows IIS (MS01-044)), DDK (an exploit to a buffer overflow vulnerability in Windows IIS (MS01-033)), and an attack against Windows Media Service (MS03-022). For each one of these attacks we created PBAs that mimic the normal traffic of five different hosts that we selected at random from the GATECH dataset. Based on the traffic of these hosts, we created several versions of the attacks by spreading the attack payload on different values of the total number of attack packets and targeting a different feature extraction method. We created PBAs that mimic the statistical distribution of n-grams with $n = 1..12$, 2_ν-grams with $\nu = 1..10$, and 2_{all}-gram which is intended to mimic all the 2_ν-grams with $\nu = 1..10$ at the same time. Overall, we generated 6,339 PBAs that aim to evade both PAYL and McPAD. It is worth noting that the main goal of the PBAs we generated is to mimic the distribution of n-grams (with different values of n) and can therefore be seen as evasion attacks against any payload-based anomaly IDS that uses n-gram analysis.

The characteristics of the ATTACKS dataset are summarized in Table 4.

Table 4
ATTACKS dataset characteristics

Type	**Attacks**	**Attack Packets**
Generic	66	205
Shell-code	11	93
CLET	96	792
PBA	6,339	71,449
Total	6,512	72,539

5.4 Experimental Results

In the first part of our experiments we show that the 2_ν-gram feature extraction technique presented in Section 4.1 is actually able to extract structural information from the payload. Afterwards, we evaluate the accuracy of McPAD in detecting *Generic Attacks*, *Shell-code Attacks* and polymorphic *CLET Attacks*. In the last part of the experiments we compare McPAD to PAYL on these three groups of attacks, and we evaluate the robustness of the two detectors in the face of advanced *Polymorphic Blending Attacks* (PBAs).

5.4.1 2_ν-gram Analysis.

We discussed in Section 4.1 how to extract the features using the 2_ν-gram technique. We also argued that the occurrence frequency of 2_ν-grams somehow "summarizes" the occurrence frequency of n-grams. This allows us to capture some byte sequence information. In order to show that the 2_ν-grams actually extract structural information from the payload, we can consider the bytes in the payload as random variables and then we can compute the relative mutual information of bytes that are ν positions apart from each other. That is, for a fixed value of ν we compute the quantity

$$RMI_{\nu,i} = \frac{I(B_i; B_{i+\nu+1})}{H(B_i)} \tag{15}$$

where $I(B_i; B_{i+\nu+1})$ is the mutual information of the bytes at position i and $(i+\nu+1)$, and $H(B_i)$ is the entropy of the bytes at position i. By computing the average for $RMI_{\nu,i}$ over the index $i = 1, .., (L-\nu-1)$, with L equal to the maximum payload length, we obtain the average relative mutual information for the 2_ν-grams along the payload. We measured this average relative mutual information on both the training and the test set varying ν from 0 to 20. The results are shown in Figure 2 for the traffic of day 1 and the merged traffic of day 2 to 5 from the GATECH dataset. It is easy to see that the amount of information extracted using the 2_ν-gram technique is maximum for $\nu = 0$ (i.e., when the 2-gram technique is used) and decreases for growing ν. However the decreasing trend is slow and the average RMI is always higher than 0.5 until $\nu = 10$. This is probably due to the fact that HTTP is a highly structured protocol.

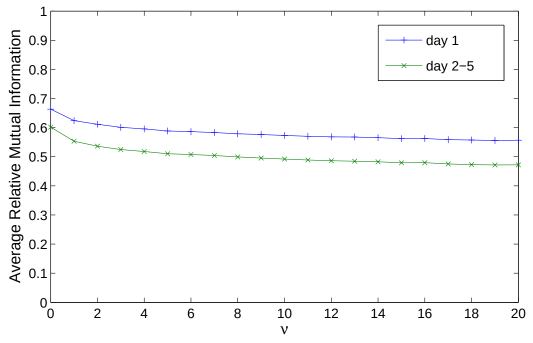

Fig. 2. Average relative mutual information for varying ν (computed on GATECH dataset).

Summing up, the 2_{ν}-gram technique indeed extracts structural information from the payload, which helps to construct accurate classifiers.

5.4.2 Validation of McPAD

Similarly to [26], we performed experiments with a combination of 11 one-class classifiers. Each classifier is trained on a different representation of normal payloads obtained using the 2_{ν}-gram technique, with $\nu = 0, .., 10$, as explained in Section 4.1. In Section 5.4.1 we showed that for $\nu \leq 10$ the relative mutual information of bytes in the payload that are ν positions apart is higher than 0.5. This means that 2_{ν}-grams with $\nu \leq 10$ actually convey structural information extracted from the payload. Also, combining 2_{ν}-grams with $\nu \leq 10$ allows us to approximate the distribution of 12-grams, which may be difficult for the attacker to mimic, as explained in [26].

In this paper, we performed separate tests using both the DARPA dataset and the GATECH dataset for the training phase and for computing the false positive rate. In both cases we used the ATTACKS dataset to estimate the detection rate. For each day of normal traffic, we trained McPAD on 80% of the traffic (the training dataset), and tuned the detection threshold (see Section 4.3) on the remaining 20% of traffic (the validation dataset) in order to obtain the desired false positive rate. We then tested the obtained classifier on 20% of the traffic from all the remaining days of traffic not involved in the training and validation process. For example, we trained and validated (for choosing the detection rate) McPAD on the first day of the DARPA dataset, and tested it on 20% of the traffic randomly sampled from days 2 to 4 and on each group of attacks in the ATTACK dataset. We repeated the same process for all the other days, thus performing a 5-fold cross validation evaluation (the DARPA dataset consists of 5 days of normal traffic). We proceeded in a similar

Table 5
DARPA dataset - summary of normalized AUC results computed over different values of k.

	Maj. Vot.	**Avg. Prob.**	**Prod. Prob.**	**Min. Prob.**	**Max. Prob.**
Generic Attacks					
MAX	0.94425	0.97837	0.97868	0.97377	0.91530
MIN	0.83100	0.87116	0.87092	0.83963	0.84533
AVG	0.89211	0.93982	0.93996	0.90682	0.87993
STDEV	0.02834	0.02672	0.02668	0.03048	0.02078
Shell-code Attacks					
MAX	0.97910	0.99990	0.99990	0.99986	0.98400
MIN	0.95410	0.95235	0.95228	0.94354	0.92836
AVG	0.96876	0.98698	0.98700	0.98537	0.96496
STDEV	0.00616	0.01684	0.01686	0.01830	0.01009
CLET Attacks					
MAX	0.99939	0.99944	0.99944	0.99969	0.99924
MIN	0.98467	0.98324	0.98318	0.97328	0.95686
AVG	0.99547	0.99639	0.99638	0.99500	0.99329
STDEV	0.00407	0.00459	0.00460	0.00717	0.00881

Table 6
GATECH dataset - summary of normalized AUC results computed over different values of k.

	Maj. Vot.	**Avg. Prob.**	**Prod. Prob.**	**Min. Prob.**	**Max. Prob.**
Generic Attacks					
MAX	0.8787	0.8978	0.8976	0.9073	0.8424
MIN	0.8054	0.8360	0.8359	0.8370	0.5822
AVG	0.8452	0.8671	0.8673	0.8752	0.7578
STDEV	0.0213	0.0169	0.0170	0.0173	0.0587
Shell-code Attacks					
MAX	0.99729	0.99991	0.99991	0.99987	0.99918
MIN	0.97904	0.98512	0.98512	0.96748	0.56770
AVG	0.98926	0.99744	0.99742	0.99492	0.92340
STDEV	0.00456	0.00330	0.00333	0.00813	0.10504
CLET Attacks					
MAX	0.99897	0.99953	0.99954	0.99970	0.99930
MIN	0.99403	0.99721	0.99722	0.99713	0.74793
AVG	0.99764	0.99829	0.99828	0.99908	0.94872
STDEV	0.00098	0.00066	0.00064	0.00070	0.08468

way for the GATECH dataset, for which we performed a 7-fold cross validation evaluation (the GATECH dataset consists of 7 days of normal traffic).

We repeated such experiments fixing the number of feature clusters k and using different values of the desired false positive rate and combination rule. Therefore, for each fixed value of k and combination rule we were able to

Table 7
DARPA dataset - Average of normalized AUC results for different values of k.

	Maj. Vot.	**Avg. Prob.**	**Prod. Prob.**	**Min. Prob.**	**Max. Prob.**
Generic Attacks					
k=10	0.87097	0.91665	0.91698	0.88028	0.87066
k=20	0.87631	0.92025	0.92013	0.90145	0.87058
k=40	0.89713	0.93291	0.93334	0.8981	0.88476
k=80	0.8954	0.95584	0.95589	0.91748	0.88482
k=160	0.92075	0.97343	0.97347	0.93677	0.8888
Shell-code Attacks					
k=10	0.96707	0.97691	0.97686	0.97572	0.96962
k=20	0.96762	0.98282	0.98281	0.98107	0.96483
k=40	0.96745	0.98256	0.98263	0.98194	0.95724
k=80	0.96958	0.99404	0.99404	0.99145	0.96713
k=160	0.97208	0.9986	0.99865	0.99668	0.966
CLET Attacks					
k=10	0.99539	0.99496	0.99495	0.99409	0.9953
k=20	0.99423	0.99489	0.99488	0.99352	0.98642
k=40	0.99412	0.99575	0.99574	0.99601	0.99424
k=80	0.9972	0.99815	0.99815	0.99558	0.99524
k=160	0.99639	0.9982	0.9982	0.99582	0.99525

compute a 5-fold and 7-fold cross validation of the normalized AUC. Table 5 and Table 6 report the maximum, minimum, average and standard deviation of the normalized cross-validation AUC computed over the different values of k for each combination rule we considered. On the other hand, Table 7 and Table 8 report the average normalized AUC obtained with cross-validation for each different value of k and different combination rule. It is easy to see that the average, product, and minimum probabilities combination rules provide the highest values of average AUC, and therefore best classification performance. Also, as we can see the average AUC reaches values very close to 1, when Shell-code and CLET Attacks are considered. As mentioned above, Shell-code and CLET Attacks carry some form of executable code in the payload. This causes the statistical distribution of byte values in the payload to be severely altered, compared to the normal distribution. We believe the presence of executable code in the payload has even a more evident effect on the distribution of 2_ν-grams. This is the reason why McPAD is so effective in detecting this kind of attacks. The Generic attacks are more difficult to detect. The reason is that this dataset contains information leakage and Denial of Service (DoS) attacks, for example, besides shell-code attacks. Information leakage and DoS attacks usually do not carry executable code, and although they include some abnormality in the payload that allows the attacker to exploit the targeted vulnerability, they do not significantly alter the distribution of byte values in the payload, compared to normal traffic.

Table 8
GATECH dataset - Average of normalized AUC results for different values of k.

	Maj. Vot.	Avg. Prob.	Prod. Prob.	Min. Prob.	Max. Prob.
Generic Attacks					
k=10	0.83501	0.86331	0.8633	0.87187	0.76765
k=20	0.8366	0.8613	0.86135	0.86882	0.7492
k=40	0.8366	0.86312	0.86407	0.87783	0.77834
k=80	0.84778	0.85948	0.8595	0.88594	0.80212
k=160	0.87016	0.8884	0.88828	0.87131	0.69164
Shell-code Attacks					
k=10	0.98632	0.99544	0.99543	0.99323	0.94105
k=20	0.98758	0.99689	0.9969	0.99361	0.94685
k=40	0.98903	0.99827	0.99826	0.99417	0.97585
k=80	0.99613	0.99874	0.99875	0.9965	0.98666
k=160	0.98723	0.99785	0.99775	0.99709	0.76661
CLET Attacks					
k=10	0.99776	0.99854	0.99854	0.99866	0.9589
k=20	0.99778	0.99839	0.99839	0.99925	0.969
k=40	0.99757	0.99815	0.99815	0.99908	0.98624
k=80	0.99773	0.99785	0.9979	0.99925	0.99669
k=160	0.99737	0.9985	0.99844	0.99913	0.83275

We also performed experiments with a number of classifiers m lower than 11. Figure 3 and Figure 4 present the results on the classification of Generic Attacks and Shell-code Attacks, respectively for different values of m between 3 and 11. The experiments were performed in the following way. We merged the 7 days of GATECH traffic, and then we split it in two parts of equal size. We used one half of the obtained dataset for training McPAD, and the second half to test the false positive rate. We used the Generic Attacks and Shell-code Attacks, respectively, to compute the detection rate. During test, for each payload in input McPAD picks m classifiers at random among the pool of 11 available classifiers (each trained with a different value of $\nu = 0..10$, as explained above), classifies the payload, and combines the obtained m outputs. Both Figure 3 and Figure 4 report the results obtained using the *minimum probability* combination rule. As we can see, the AUC slightly decreases for decreasing values of the number of feature clusters k. Also, the AUC decreases with the number of combined classifiers m. However, it is worth noting that in Figure 4 the AUC is always higher than 0.97, and for $k = 160$ the AUC is always higher than 0.985 even when $m = 3$. This confirms that McPAD is very good at detecting Shell-code Attacks, even when the number of one-class classifiers in the ensemble is low. On the other hand, Figure 4 shows that McPAD suffers more when detecting Generic Attacks for low values of m and k, although the AUC stays above 0.8 when $k = 160$ is used. In Section 5.4.4 we will discuss how the number of classifiers m and the value of k impact the average computational cost per payload.

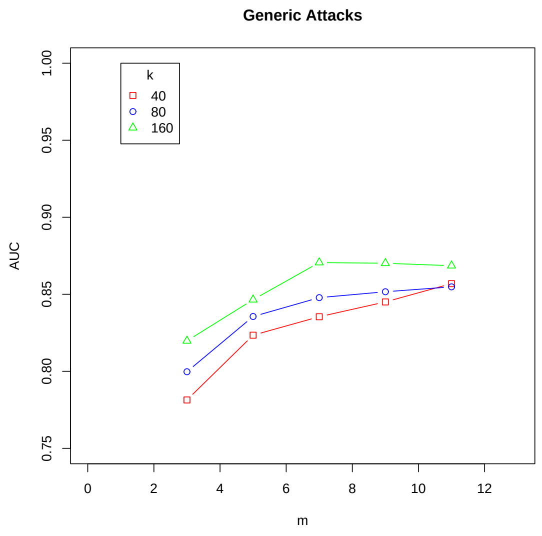

Fig. 3. Generic Attacks detection - AUC obtained for different values of the number of the feature clusters k, and number of combined classifiers m. The combination rule used was *minimum probability*.

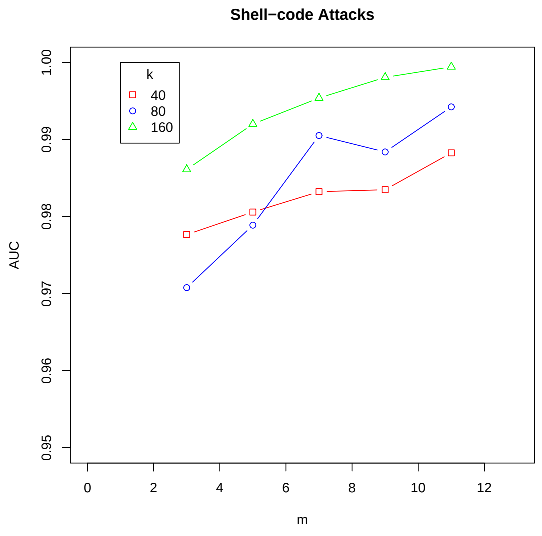

Fig. 4. Shell-code Attacks detection - AUC obtained for different values of the number of the feature clusters k, and number of combined classifiers m. The combination rule used was *minimum probability*.

5.4.3 Comparison between McPAD and PAYL

In this section we present the results of the comparison between McPAD and PAYL. In order to compare the two anomaly detectors, we proceeded this way: we first merged the 5 days of traffic of the DARPA dataset and than we randomly split the obtained traffic in two portions, 50% reserved for training purposes and 50% for testing. We did the same for the GATECH dataset, i.e., we merged the 7 days of traffic and then we randomly split it in two parts of roughly the same size. First, we trained both McPAD and PAYL on the first half of the DARPA dataset, and then we tested both the detectors on the second half of the DARPA dataset and the entire ATTACKS dataset. We repeated the same procedure using the GATECH dataset. For the sake of brevity, in the following we only present the results obtained using the GATECH dataset. The results on the DARPA dataset are similar. We set the parameters of McPAD to $k = 160$, a desired false positive rate of 1% for each single one-class SVM classifier, and the *maximum probability* combination rule. We chose this configuration of McPAD because it provided slightly better results compared to other configurations we tried for the detection of polymorphic blending attacks at very low false positives, while maintaining also good results for the detection of shell-code attacks and polymorphic CLET attacks at very low false positives as well. We varied the detection threshold on the output of the combination of classifiers, so to vary the trade-off between false positives and detection rate, thus allowing us to draw the ROC curve.

Figure 5 and Figure 6 report the results obtained with PAYL and McPAD, respectively. In this case, the test dataset consisted of the second half of the GATECH traffic, which we use to compute the false positives, and the *Generic Attacks*, *Shell-code Attacks*, and *CLET Attacks*, on which we computed the detection rate. The three curves in the graph reflect the obtained results. It is easy to see that for all of the three groups of attacks, the detection rate of PAYL rapidly decreases for a false positive rate below $5 \cdot 10^{-3}$. On the other hand, McPAD is able to detect the three groups of attacks even at a false positive rate of 10^{-5}. In particular, McPAD is able to detect shell-code attacks and polymorphic shell-code attacks generated using CLET very well, with a detection rate around 90% and above even at very low false positive rates.

Figure 7 and Figure 8 show the results obtained on several Polymorphic Blending Attacks (PBAs) derived from the Code-Red worm. Although we generated a high number of polymorphic attacks, here we only report part of the results, for the sake of brevity. Both figures report the ROC curves obtained using n-gram Code-Red attacks with $n = 1, 2, 4, 12$, and 2-all-gram attacks (see Section 5.3) constructed using either 5 or 10 overall attack packets (reported at the end of the attacks' name in the graph legend). Intuitively, the larger the number of packets, the larger the space available for the attacker to better mimic the distribution of normal traffic [14, 13], and thus the more difficult it is to detect the attack. It is easy to see that while the detection rate of PAYL rapidly drops at a false positive rate around 10^{-3}, in some cases McPAD is able to "push" the ROC curves farther to the left (lower false positives). This shows that McPAD is more robust than PAYL for PBA instances that are spread over a limited number of packets. However, when the attack is spread over a

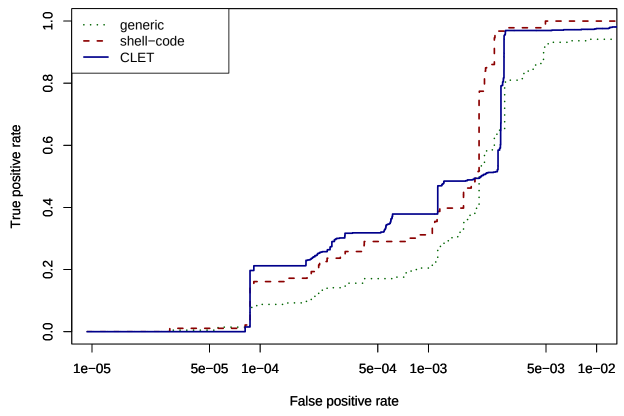

Fig. 5. PAYL - ROC curves for Generic, Shell-code, and CLET attacks

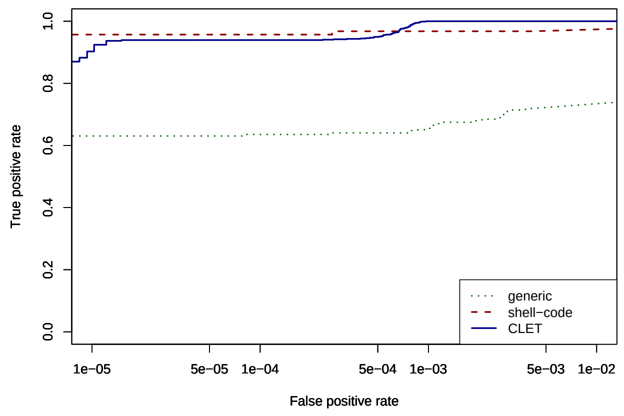

Fig. 6. McPAD - ROC curves for Generic, Shell-code, and CLET attacks

large number of packets (e.g. 10 in the case of the Code-Red PBA), neither PAYL nor McPAD is able to detect the attack at very low false positives.

Figure 9 and Figure 10 show similar results for PBAs derived from the DDK attack using either 3 or 5 packets., whereas Figure 11 and Figure 12 show the results obtained with PBAs derived from the WMS attack using either 1 or 3 packets.

5.4.4 Computational Cost Analysis

In this section we discuss the experimental results regarding the performance of McPAD and PAYL in terms of average computational cost per payload. We performed the experiments on a machine equipped with a 2GHz Dual Core

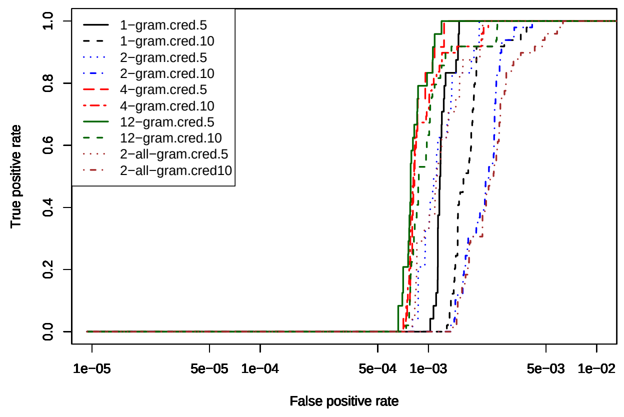

Fig. 7. PAYL - ROC curves for Code-Red PBA attacks (the string "cred" in the legend stands for "code-red").

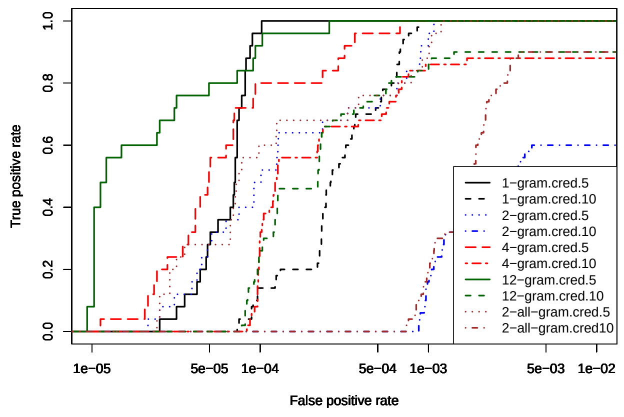

Fig. 8. McPAD - ROC curves for Code-Red PBA attacks (the string "cred" in the legend stands for "code-red").

AMD Opteron$^{\text{TM}}$ Processor and 8GB of RAM, although both McPAD and PAYL used only one CPU core at a time, and always less than 4GB of RAM.

Table 9 and Table 10 report the results obtained with PAYL and McPAD, respectively, on both the DARPA and GATECH datasets. The numbers between parenthesis in the first column in both tables represet the number of payloads in each test dataset. We filtered out all the packets which did not carry a TCP payload (e.g., we did not count the time spent on SYN, and FIN packets). The average time per payload is reported in terms of milliseconds. In Table 10 we report the results obtained considering only a few possible parameter configurations for McPAD, for the sake of brevity. As discussed above, k represents the number of feature clusters, whereas FP represents the

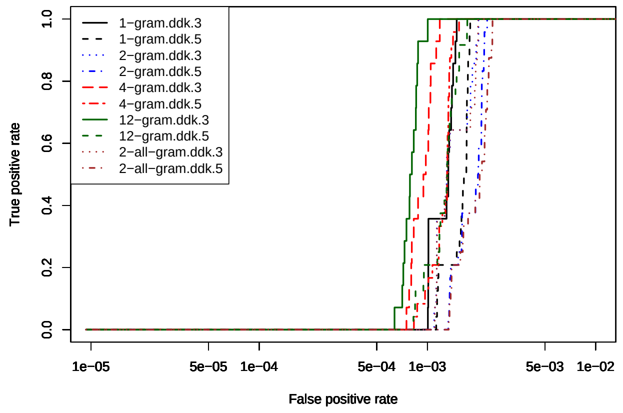

Fig. 9. PAYL - ROC curves for DDK PBA attacks.

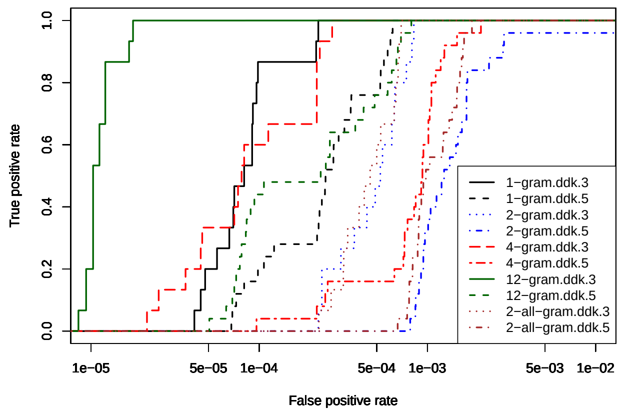

Fig. 10. McPAD - ROC curves for DDK PBA attacks.

"desired" false positive rate which we chose to train the SVM models and tune the detection threshold on the output of the combination rules. m represents the number of combined models. We experimented with $m = 11$, which refers to the combination of all the SVM models constructed on different values of the parameter $\nu = 0..10$. Also, we experimented with $m = 3$ by picking 3 different models out of the available 11 SVM models at random in order to classify each payload. In other words, during the test phase, for each payload McPAD picks 3 of the 11 SVM models at random, classifies the payload using only the chosen 3 models, and combins the obtained 3 outputs to make the final decision about whether the payload under test is anomalous or not. We found that this approach decreases the average computation cost per payload while maintaining a high detection rate at very low false positive rates in most scenarios, as shown in Figure 13 and Figure 14 (these two figures were obtained using the same configuration of McPAD used to plot Figure 6 and Figure 8, with the only difference that we combined only 3 classifiers, instead

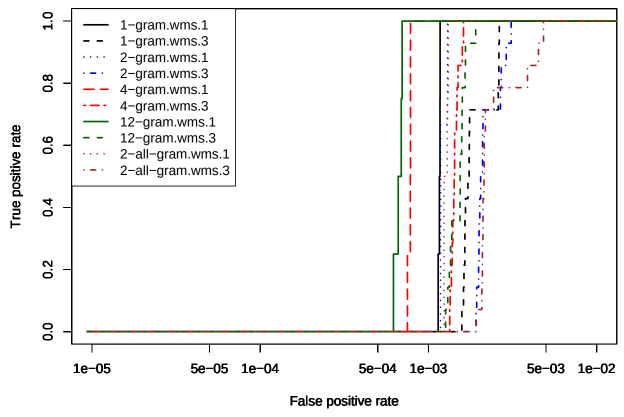

Fig. 11. PAYL - ROC curves for WMS PBA attacks.

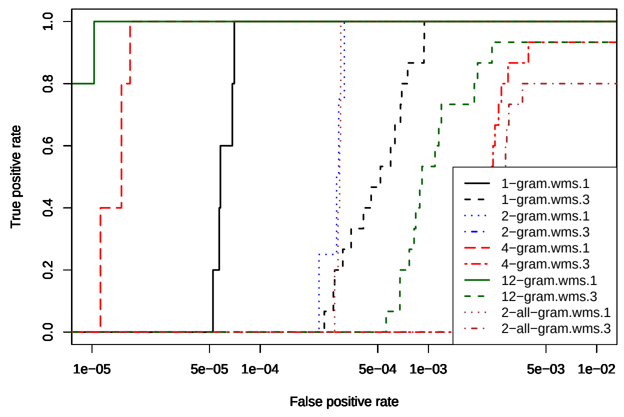

Fig. 12. McPAD - ROC curves for WMS PBA attacks.

of 11).

It is easy to see that the results reported for McPAD in Table 10 are consistent with the computational complexity analysis in Section 4.4, in particular for the results on the GATECH dataset. The difference between the results on the DARPA and GATECH datasets are due to the fact that for a fixed “desired” false positive rate the number of support vectors grows with the size of the training dataset. As the GATECH training dataset is larger than the DARPA training dataset, the SVM models used to classify GATECH traffic will have more support vectors (see Section 4.4), and therefore a higher average cost per test payload.

The performance of PAYL in terms of average computational cost per payload is much better than our McPAD software. However, it is worth noting that our software is a proof-of-concept Java implementation of the algorithms described in this paper, and that both LibSVM, on which McPAD is based,

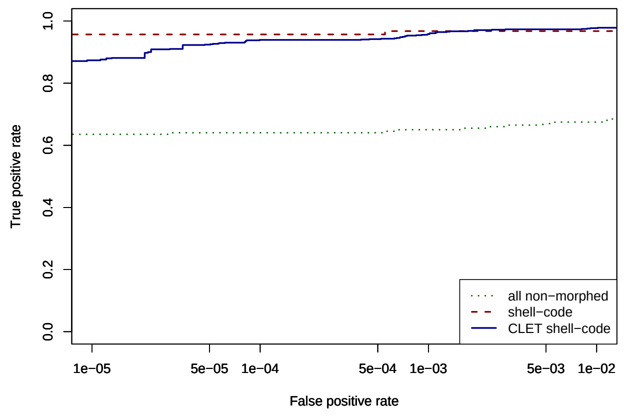

Fig. 13. McPAD - ROC curves for Generic, Shell-code, and CLET attacks. The ROC refers to classification results obtained by combining 3 classifiers chosen at random among 11 one-class classifiers.

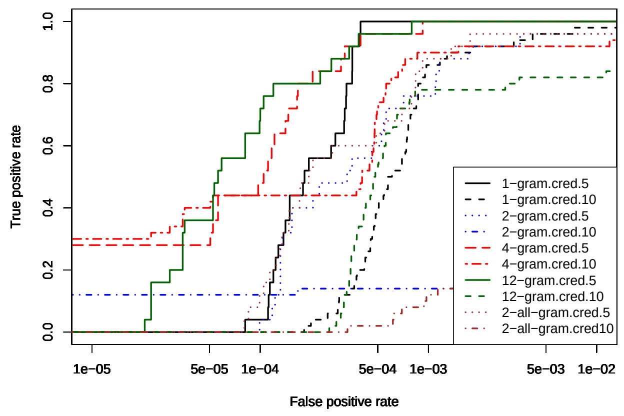

Fig. 14. McPAD - ROC curves for Code-Red PBA attacks. The ROC refers to classification results obtained by combining 3 classifiers chosen at random among 11 one-class classifiers.

and McPAD itself are not optimized. In particular, the classification of a payload using a given one-class SVM can be easily parallelized. This is because the distance between the pattern vector representing the payload and each support vector in Equation (3) can be computed independently. The results can then be summed up and compared to the threshold ρ in order to compute the probability $p(\mathbf{x}|\omega_t)$ in Equation (4). Also, the classification results of each single one-class SVM can be computed independently and then combined using one of the combination rules discussed in Section 3.3 (e.g., average of probabilities).

Table 9
PAYL's average processing time per payload. The number between parenthesis represents the number of payloads in the test dataset.

DARPA (137,997)	0.039ms
GATECH (1,068,429)	0.032ms

Table 10
McPAD's average processing time per payload. The number between parenthesis in the first column represents the number of payloads in the test dataset.

		FP=0.001	**FP=0.001**	**FP=0.01**	**FP=0.01**
		m=3	**m=11**	**m=3**	**m=11**
DARPA (137,997)	**k=10**	3.07ms	10.96ms	3.16ms	11.04ms
	k=40	3.04ms	11.02ms	3.11ms	11.31ms
	k=160	3.13ms	10.92ms	3.81ms	13.39ms
GATECH (1,068,429)	**k=10**	4.28	16.23ms	4.94ms	17.53ms
	k=40	4.16ms	15.92ms	6.14ms	21.93ms
	k=160	4.95ms	17.11ms	10.49ms	38.45ms

Another approach that can be used to optimized the performance of our IDS, is to use it as a *second-stage classifier*. For example, we could use PAYL configured with a detection threshold that forces it to generated a high percentage of FP rate, say 5%, for example. In turn, this high percentage of "desired" false positives forces PAYL to classify as normal only those payloads that have a very low anomaly score (as computed by PAYL itself), and for which we are therefore very confident that they do not carry an attack (as we can see from Figure 5, 7, 9, and 11, PAYL has a high detection rate at false positive rates higher than 1%). Only those payloads that are classified as anomalous by PAYL may then be fed to McPAD, which should be configured in order to accept only a small percentage of false positives, say 0.01%. As McPAD has a higher detection rate than PAYL for very low false positive rates (in particular for shell-code and CLET attacks), the overall result is a two-stage IDS characterized by high detection rate at very low false positive rate. At the same time, as in average only 5% of the payloads will undergo "precise scrutiny" by McPAD, the computational cost per payload of the entire two-stage classifier will also be much lower in average, compared to using McPAD by itself.

It is also worth noting that techniques like load-balancing can be employed. For example, it is typical to have more than one web server offering the same service, for both efficiency and fault tolerance reasons. Putting an instance of our IDS in front of each of a pool of N Web servers in load balance would decrease the traffic crossing each instance of McPAD, thus multiplying the overall traffic that can be handled by the provider of the web services by N. Although this solution may be financially more expensive, we believe that in a number of practical scenarios the increase in security provided by our IDS would generate a positive *return of investment*. This reasoning is supported by the analysis presented in Section 5.5, where we show that PAYL has a much lower Bayesian detection rate compared to McPAD.

5.5 Bayesian Detection Rate

The experimental results show that McPAD is able to detect attacks even at very low false positive rates. This is particularly true for shell-code attacks and polymorphic attacks created using morphing engines such as CLET [8]. On the other hand the detection rate of PAYL quickly drops to zero at low false positive rates. This is an important result, because being able to maintain a high detection rate with very low false positives greatly improves the Bayesian detection rate $P(Intrusion|Alarm)$ [2], i.e., the probability of having an intrusion given that the IDS issued an allarm. The Bayesian detection rate is defined as:

$$P(I|A) = \frac{P(A|I)P(I)}{P(A|I)P(I) + P(A|\bar{I})P(\bar{I})} \tag{16}$$

where we used I for *Intrusion*, A for *Alarm*, and $\bar{I}$ for *Not Intrusion*. It is easy to see that the detection rate can be viewed as an estimate of $P(A|I)$, whereas the false positive rate is an estimate of $P(A|\bar{I})$. Because intrusion attempts are not frequent and the number of attack packets usually represents a very small fraction of the traffic [2], the probability $P(I) = 1 - P(\bar{I})$ is usually very small. As a consequence, the Bayesian detection rate $P(I|A)$ will be dominated by $P(A|\bar{I})$, i.e., the false positive rate [2]. If the false positive rate is not the same order of magnitude as the *a priori* probability of having an intrusion, $P(I)$, or lower, the denominator in Equation (16) will be governed by the false positive rate $P(A|\bar{I})$. This confirms how the Bayesian detection rate is greatly influenced by the false positive rate, and shows us that in order to have a high Bayesian detection rate we need to lower the false positive rate as much as possible.

In [2], Axelsson presented a realistic example in which he showed that in order to have a significant Bayesian detection rate we need to reduce the false positive rate to around 10^{-5}, while maintaining a relatively high detection rate. It is easy to see from Figure 8 that McPAD achieves this goal in particular in case of shell-code attacks and polymorphic CLET attacks. Following Axelsson's example [2], assume $P(I) = 1 - P(\bar{I}) = 2 \cdot 10^{-5}$. McPAD has a detection rate around 95% for shell-code attack packets, at a false positive rate of 10^{-5}.

[2] This may not be true in case of large-scale distributed attacks. However, such attacks are rare and limited to a defined, usually relatively short, period of time.

Therefore, $P(A|I) = 0.95$, and $P(A|\bar{I}) = 10^{-5}$. In this case the Bayesian detection rate is $P(I|A) = 0.65$. On the other hand, PAYL has detection rate equal to zero at a false positive rate of 10^{-5}, and therefore the Bayesian detection rate is zero. If we consider that the detection rate of PAYL is around 0.32 at a false positive rate of 10^{-3}, in this case we have a Bayesian detection rate for PAYL equal to $P(I|A) = 0.006$. Even if we had 100% detection rate at 10^{-3}, the Bayesian detection would be only $P(I|A) \simeq 0.02$. This confirms that we need to maintain a high detection rate at very low false positive rates, in order to increase the Bayesian detection rate. We showed that McPAD meets this requirement.

6 Conclusion and Future Work

In this paper we presented McPAD (Multiple-Classifier Payload-based Anomaly Detector), a new accurate payload-based anomaly detection systems that consists of an ensemble of one-class classifiers. We showed that our anomaly detector has a high detection accuracy for shell-code attacks and polymorphic attacks generated using the morphing engine CLET [8]. This holds true even in the case of very low false positive rates. We showed that this is fundamental in order to obtained a significant Bayesian detection rate $P(Intrusion|Alarm)$, i.e., the probability of having an intrusion attempt given that the Intrusion Detection System (IDS) raised an alarm. We also compared McPAD to PAYL [35], and we showed that PAYL is not able to detect network attacks at very low false positives rates. Therefore, we showed that McPAD provides a much higher Bayesian detection rate, compared to PAYL.

Furthermore, we experimented with advanced *polymorphic blending attacks* and we showed that in some cases even in the presence of such sophisticated attacks and for low false positive rates our IDS still has a relatively high detection rate. On the other hand, PAYL has a detection rate for polymorphic blending attacks close to zero, at very low false positive rates. In case of advanced Polymorphic Blending Attacks (PBAs) McPAD provides more robustness compared to PAYL, although it does not perform well when the attacker tries to spread the PBA over several attack packets. We intend to address the problem of further improving the robustness of McPAD against PBAs in our future work. In particular, we will study the effectiveness of extracting additional features for modeling the normal traffic. For example, an additional feature may be the total length of TCP flows, which may allow us to detect attempts of spreading polymorphic blending attacks over a large number of packets in order to evade detection. Also, adding semantic information about the protocol may help improving the model of normal traffic. For example, it may be possible to train different models of normal traffic for GET and POST requests, in case of the HTTP traffic. This information is easy to extract and does not add much overhead to the feature extraction process. On the other hand, having distinct models for different types of requests may make modeling of the normal traffic more precise, and the classification results more accurate.

References

[1] I. Arce. The shellcode generation. *IEEE Security and Privacy*, 2(5):72–76, 2004.

[2] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 1–7, 1999.

[3] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[4] R. Brunelli and D. Falavigna. Person identification using multiple cues. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(10):955–966, 1995.

[5] R. Chinchani and E.V.D. Berg. A fast static analysis approach to detect exploit code inside network flows. In *Recent Advances in Intrusion Detection (RAID)*, 2005.

[6] L. P. Cordella, A. Limongiello, and C. Sansone. Network intrusion detection by a multi-stage classification system. In *Multiple Classifier Systems (MCS)*, pages 324–333, 2004.

[7] C. Cortes and M. Mohri. Confidence intervals for the area under the roc curve. In *NIPS 2004: Advances in Neural Information Processing Systems*, 2004.

[8] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. Polymorphic shellcode engine using spectrum analysis. *Phrack Issue 0x3d*, 2003.

[9] I. S. Dhillon, S. Mallela, and R. Kumar. A divisive information-theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3:1265–1287, 2003.

[10] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems (MCS)*, 2000.

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2000.

[12] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In D. Barbara and S. Jajodia, editors, *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[13] P. Fogla and W. Lee. Evading network anomaly detection systems: formal reasoning and practical techniques. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 59–68,

2006.

[14] P. Fogla, M. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee. Polymorphic blending attack. In *USENIX Security Symposium*, 2006.

[15] G. Giacinto, R. Perdisci, M. Del Rio, and F. Roli. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion*, 9(1):69–82, 2008.

[16] G. Giacinto, F. Roli, and L. Didaci. Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern Recognition Letters*, 24(12):1795–1803, 2003.

[17] K. L. Ingham and H. Inoue. Comparing anomaly detection techniques for HTTP. In *Recent Advances in Intrusion Detection (RAID)*, 2007.

[18] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

[19] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *ACM Symposium on Applied Computing (SAC)*, 2002.

[20] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms.* Wiley, 2004.

[21] E. Leopold and J. Kindermann. Text categorization with support vector machines. How to represent texts in input space? *Machine Learning*, 46:423–444, 2002.

[22] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

[23] M. V. Mahoney and P. K. Chan. An analysis of the 1999 darpa lincoln laboratory evaluation data for network anomaly detection. In *Recent Advances in Intrusion Detection (RAID)*, 2003.

[24] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[25] J. McHugh, A. Christie, and J. Allen. Defending yourself: The role of intrusion detection systems. *IEEE Software*, pages 42–51, Sept./Oct. 2000.

[26] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 488–498, 2006.

[27] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *ACM CSS Workshop on Data Mining Applied to Security*, 2001.

[28] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and RC Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443–1471, 2001.

[29] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002.

[30] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. On the infeasibility of modeling polymorphic shellcode. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[31] D. M. J. Tax. *One-Class Classification, Concept Learning in the Absence of Counter Examples.* PhD thesis, Delft University of Technology, Delft, Netherland, 2001.

[32] D. M. J. Tax and R. P. W. Duin. Combining one-class classifiers. In *Multiple Classifier Systems (MCS)*, 2001.

[33] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Recent Advances in Intrusion Detection (RAID)*, 2002.

[34] V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[35] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection (RAID)*, 2004.

[36] K. Wang and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *Recent Advances in Intrusion Detection (RAID)*, 2005.

[37] K. Wang and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection (RAID)*, 2006.