# HMMPayl: an Intrusion Detection System based on Hidden Markov Models

Davide Ariu[a], Roberto Tronci[a], Giorgio Giacinto[a]

[a] *Department of Electric and Electronic Engineering, University of Cagliari*
*Piazza d'Armi, 09123 Cagliari, Italy*

**Abstract**

Nowadays the security of Web applications is one of the key topics in Computer Security. Among all the solutions that have been proposed so far, the analysis of the HTTP payload at the byte level has proven to be effective as it does not require the detailed knowledge of the applications running on the Web server. The solutions proposed in the literature actually achieved good results for the detection rate, while there is still room for reducing the false positive rate.

To this end, in this paper we propose *HMMPayl*, an IDS where the payload is represented as a sequence of bytes, and the analysis is performed using Hidden Markov Models (HMM). The algorithm we propose for feature extraction and the joint use of HMM guarantee the same expressive power of $n-gram$ analysis, while allowing to overcome its computational complexity. In addition, we designed *HMMPayl* following the Multiple Classifiers System paradigm to provide for a better classification accuracy, to increase the difficulty of evading the IDS, and to mitigate the weaknesses due to a non optimal choice of HMM parameters. Experimental results, obtained both on public and private datasets, show that the analysis performed by *HMMPayl* is particularly effective against the most frequent attacks toward Web applications (such as XSS and SQL-Injection). In particular, for a fixed false positive rate, *HMMPayl* achieves a higher detection rate respect to previously proposed approaches it has been compared with.

*Keywords:* Network Intrusion Detection, Anomaly Detection, Multiple classifiers, Hidden Markov Models, Payload Analysis

*Email addresses:* davide.ariu@diee.unica.it (Davide Ariu), roberto.tronci@diee.unica.it (Roberto Tronci), giacinto@diee.unica.it (Giorgio Giacinto)

## 1. Introduction

Nowadays Web-based services and social networking platforms are quite common, and their number is still increasing as the Web-based architecture is the most frequently used in software deployments. The results of a recent study by the X-Force team show that approximately 50% of vulnerabilities discovered during 2009 affected Web applications [1]. In consequence of this, the security of Web applications is a key topic in computer security.

The protection of Web applications is challenging, because they are in general large, complex, highly customized and often created by programmers with poor security background. On the other hand, a requirement that a tool to protect Web applications is desired to meet is being as autonomous as possible, i.e., it should not require extensive administration overhead. Several hardware and software solutions have been developed, and are available on the market. Among these, Web Application Firewalls are one of the most frequently used protection tools. Typically, they rely on a set of rules written by the administrator, who therefore must have an in-depth knowledge of the applications to be protected. Even if there are solutions as ModSecurity [2] that offer automated rule update functionalities, tools that rely only on rule-based approaches do not seem to guarantee sufficient protection to Web applications. The main reason is that zero-day attacks are particularly critical for Web applications, because exploiting a vulnerability in an application with a large number of users might allow to quickly infect a large number of victim clients (e.g., the so-called drive by download attacks). In our opinion, anomaly detection approaches allow addressing the limitation of rule-based systems, thus providing for an effective solution.

Anomaly-based systems rely on a model of the normal behavior of Web applications. We share the definition of "normal behavior" provided by Maggi et. al. in [3]: *the term normal behavior generally refers to a set of characteristics (e.g., the distribution of the characters of string parameters, the mean and standard deviation of the values of integer parameters) extracted from HTTP messages that are observed during normal operation.* Starting from this definition, it is quite straightforward defining "anomalous" all those behaviors that significantly deviate from the statistical model of the normal activity. Obviously this allows anomaly based IDS to fight off also zero-day attacks. Initially, the main obstacle to the large scale deployment of anomaly based solutions has been the too high false positive rate, as not all the detected anomalies are actually related to attack at-

tempts. Nowadays anomaly detection mechanisms are also deployed within some commercial products [4, 5, 6].

Several works that focused on anomaly-based *high speed* classification, proposed the use of simple statistics on the *application-layer payload* to characterize the normal behavior of Web applications [7, 8, 9, 10, 11, 12]. The payload is the data portion of a network packet, that is the portion of the network packet which carries the HTTP message. According to `RFC 2616`, the HTTP payload contains the Request-Line plus the Request-Header fields used by the client to pass additional information to the server [13]. In this work, we are interested in the analysis of the HTTP payload. The HTTP payload analysis offers the great advantage that the security of a Web application can be guaranteed by simply analyzing the network traffic incoming to the Web server, without a detailed knowledge of how the Web application is implemented and works.

The statistical models of the payload based on the byte distribution proposed in [9, 10, 12] do not allow for an in-depth and accurate analysis. In particular the $n$-gram-analysis proposed in [9] becomes quickly unfeasible as the value of $n$ increases, whereas the $2-\nu$-gram-analysis proposed in [12] provides only an approximate representation of the payload. From a practical perspective, this might lead to a too high false positive rate [9] or to a low detection rate [9, 10, 12] for those attacks that do not affect the payload structure too much (e.g. XSS and SQL-Injection).

In order to make this more clear, let us consider the two examples of attack in figure 1 and 2.

```
HEAD / aaaaaaaaa ... aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Figure 1: Long Request Buffer Overflow attack. **Bugtraq: 5136**

The attack in figure 1 contains a sequence of 4,096 “a” (we omitted a large number of them for typesetting constraints) which overfills a buffer producing a Denial of Service. The byte distributions in this packet is definitely anomalous with respect to that of a normal packet in consequence of the pronounced peak corresponding to the “a” character. Any statistical model of the payload would spot the anomaly allowing to detect the attack. If the attack also injects an executable code in the victim machine, that code would appear in the packet after the sequence of “a”. That would be an even easier situation for the IDS, as the injected code would also contain non-ASCII characters. In general, the attacks that aim to overflow a buffer

in the victim machine contain just the request URI. Consequently they lack the sequence of HTTP headers (and corresponding values) that are present in normal payloads.

```
GET /d/winnt/system32/cmd.exe?/c+dir HTTP/1.0
Host: www
Connnection: close
```

Figure 2: URL decoding error attack. **Microsoft: MS01-020**

The attack in figure 2 exploits a fault of the Web server in decoding the URL. In consequence of it the Web server makes an error in decoding the URL and a Denial of Service is caused. It is quite evident that the packet that carries an attack like this looks definitely more similar to a normal one with respect to that in figure 1. More in general, payload statistics of attacks similar to the above *URL decoding attack* example, are not so far from those of a normal payload. Thus it follows that a simple statistical model of the payload is sufficient to spot attacks similar to that in figure 1, while a more detailed representation is necessary to detect attacks such as that in figure 2. Previously proposed approaches suffered in detecting attacks such as that in figure 2 in consequence of a poor representation of the payload [9, 10, 12].

In this paper we address the above problems in payload analysis by proposing a novel solution where the HTTP payload is analyzed using Hidden Markov Models. The proposed system, named *HMMPayl*, performs payload processing in three steps as shown in figure 3. First of all, the algorithm we propose for *Feature Extraction* (step 1) allows the HMM to produce an effective statistical model which is sensitive to the "details" of the attacks (e.g., the bytes that have a particular value). Since HMM are particularly robust to noise, their use during the *Pattern Analysis* phase (step 2) guarantees to have a system which is robust to the presence of attacks (i.e., noise) in the training set. In the *Classification* phase (step 3) we adopted a Multiple Classifier System approach, in order to improve both the accuracy and the difficulty of evading the IDS. Besides, the MCS paradigm guarantees that the weaknesses of classifiers due to a sub-optimal choice of initial parameters are mitigated. In the same step, we also evaluated the performance of a classifiers selection approach that provides an upper bound on accuracy and may suggest some guidelines to exploit the combination of classifiers [14].

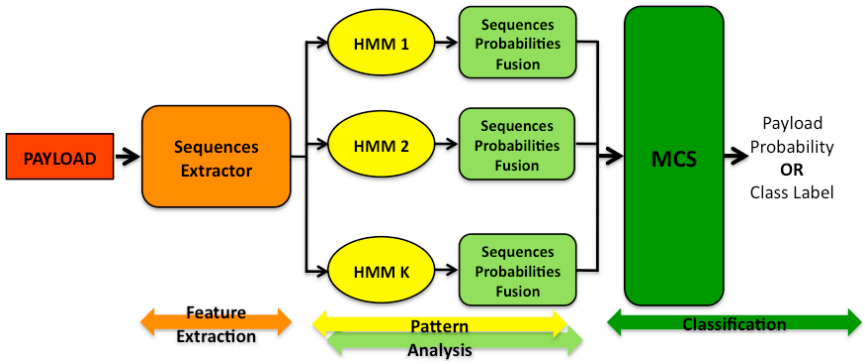


Figure 3: A simplified scheme of *HMMPayl*.

We deeply tested *HMMPayl* on several datasets of legitimate traffic and attacks. In addition we performed an in-depth comparison with some other IDS based on payload analysis that have been recently proposed in the literature. In particular we compared *HMMPayl* with two network-based IDS, namely *McPAD*, and *Spectrogram* [12, 15], and a host-based IDS, namely HMM-Web [16]. Other well known IDS such as *PAYL* and *Anagram* have not been included in this comparison since they are not as effective as more recent solutions such as *McPAD* and *Spectrogram* [12, 15]. Experimental results show that *HMMPayl* is able to achieve good results with respect to the other IDS it has been compared to.

The rest of the paper is organized as follows. In section 2 we summarize the most relevant related works, whereas in section 3 we give some background on Hidden Markov Models, classifier fusion and selection. In section 4 a detailed description of *HMMPayl* is provided. In sections 5 and 6 we describe respectively the experimental setup and results. In section 7 we evaluate the computational cost of *HMMPayl*. We then draw our conclusions in section 8.

## 2. Related Work

A number of payload-based anomaly IDS have been proposed in the literature. In the following we will briefly review the main characteristics of previously proposed solutions.

In [7] Kruegel et al. describe a service-specific Intrusion Detection System. They combine the type, length, and payload byte distribution of the service requests to build a statistical model of normal traffic that is used to compute an anomaly score.

*Netad* monitors the first 48 bytes of IP packets [8]. 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.

In *PAYL*, intrusions are detected by analyzing the distribution of bytes inside the HTTP payload [9]. In particular, the analysis performed by *PAYL* is known as $n$-gram-analysis, and has been originally used for text-classification [17]. By $n$-gram-analysis, a payload is represented through a vector containing the relative frequencies of $n$-grams, that are sequences of contiguous bytes. If $n = 1$, the histogram of the byte distribution in the payload is drawn. The relative position of different bytes inside the payload is not taken into account, so that the structure of the payload is not modeled. To model the structure of the payload, a value of $n \geq 2$ should be considered. Unfortunately, the representation of the payload by $n$-gram analysis generate a feature space of size $256^n$. It is easy to see that as the value of $n$ increases the problem becomes quickly intractable, and we would never see enough training data to properly fit a full $n$-gram distribution. This is the reason why in a real scenario a value of $n$ greater than 2 can't be used. Another element that must be considered is that the distribution of $n$-grams changes with the length of the payload. For example, large payloads are more likely to contain non-printable characters which are typical of media formats and binary representations. To take into account different payload lengths, *PAYL* employs a different model for each different length range.

The developer of *PAYL* presented an improved version in [10]. In particular, the new version builds a number of models for each packet length, and performs inbound and outbound traffic correlation to detect the propagation of worms.

In [11] Wang *et al.* propose a solution called *Anagram* where $n$-grams are extracted from both legitimate and intrusive traffic. *Anagram* stores all the $n$-grams extracted from the normal traffic, and trains a Bloom filter. Another Bloom filter stores the $n$-grams extracted from known malicious packets. At detection time, the packet is scored on the basis of the number of unobserved $n$-grams. The number of malicious $n$-grams is used to weight the score. In *Anagram* the problem is represented as that of distinguishing between two classes of patterns, namely the normal class and the malicious class. It is worth noting that in our approach (*HMMPayl*) we adopt a one-class approach, as we do not consider anomalous patterns for training. So *Anagram* is fundamentally different from *HMMPayl*.

In [12] Perdisci *et al.* propose an IDS called *McPAD*, which implements a feature extraction algorithm that can be considered as an approximation

to the $n$-gram analysis. *McPAD* models the payload of normal traffic using a 2-$\nu$-gram analysis, where relative frequencies of pairs of bytes that are from 0 to $\nu$ positions away from each other are calculated. The payload is thus represented in $\nu$+1 different feature spaces, and a one-class classifier is trained on each feature space. The main advantage of *McPAD* with respect to *PAYL* is that it is possible to approximate an $n$-gram analysis with $n$ bigger than 2 by combining the $\nu$+1 classifiers trained in different feature spaces of size $256^2$. For example, if $\nu$ is equal to 10, it is possible to approximate an $n$-gram analysis with $n = 12$, that would require a feature space of size $256^{12}$. The drawbacks of this approach are that *McPAD* only approximates a full $n$-gram analysis, and that a different classifier for each feature space must be trained.

Markovian models and Hidden Markov Models have been used for modeling computer security problems only recently, whereas they have been originally used in applications such as speech recognition [18], handwritten text recognition [19], and biological sequence analysis [20]. In the field of computer security, HMM have been largely employed in host based Intrusion Detection. The seminal work in this direction is that of Warrender *et al.*, where HMM are used to model system call sequences [21]. In [22] HMM have been used to model privilege flows, while in [23] Gao *et al.* propose to use HMM for computing a behavioral distance between processes.

Markovian Models have been also proposed for the development of tools to protect Web Applications. In [24] the authors proposed a framework to detect attacks against Web servers and Web-based applications. Attack detection is performed by multiple model, including a Markov Model that models the request URI.

*Spectrogram* is another sensor that aims at detecting Web-layer code injection attacks by operating above the packet layer [15]. *Spectrogram* uses a mixture of Markov chains to address the "curse of dimensionality" problem arising from the need for a large value of $n$ in the $n$-gram analysis. In particular, *Spectrogram* models the portion of the request URI containing the sequence of attributes (and values) received as input by a Web application. *Spectrogram*, as well as *HMMPayl*, does not take care of the details of the Web applications that are hosted by the Web server. For instance it is not important to know how many applications are running on the Web server or which kind of input each application receives. This is a consequence of the architecture of these IDS, since both solutions are based on a single model that is used to classify all the traffic toward the Web server. This model is a mixture of Markov chains for *Spectrogram*, and an ensemble of HMM for *HMMPayl*.

It is worth noting that the payload analysis performed by the above IDS is carried out by considering the raw sequence of bytes. On the other hand, a model of the payload can be attained by taking into account the detailed knowledge of the applications hosted by the Web server. In this case, the analysis of the payload takes into account its semantic. This approach has been used in HMM-Web that is made up of multiple modules, each module being related to a particular application in the Web server [16]. This detailed model allows HMM-Web being particularly accurate in detecting Web-layer code injection attacks. On the other hand, such an IDS is completely blind with respect to attacks that do not exploit Web applications vulnerabilities (e.g. attacks that exploit Web server vulnerabilities). Another drawback is the fact that even small modifications of the applications hosted by the Web server, require the IDS being re-trained.

One of the reasons which initially motivated the ideas beyond the development of *HMMPayl* was the consideration that HMM, as well as $n$-grams, have been widely used in text classification [19]. In particular, it has been shown that HMM and $n$-grams are rooted in the same theoretical model [25]. In particular, HMM as well as $n$-grams can be described using a stochastic finite state automata (sFSA) [15, 20, 26].

In spite of the same expressive power, HMM offers a great advantage with respect to $n$-grams in modeling sequences of bytes, While $n$-grams produce feature spaces of size $256^n$, HMM can process sequences of any size, without differences in computational complexity. Thus we expect to produce an effective representation of the payload. It is worth noting that a similar approach has been proposed in *Spectrogram* [15]. This work, which has been developed independently from *Spectrogram*, propose an original representation of the payload, and a multiple classifier architecture.

## 3. Theoretical Background

### *3.1. Hidden Markov Models*

*Hidden Markov Models* represent a very useful tool to model data-sequences, and to capture the underlying structure of a set of strings of symbols. HMM is a stateful model, where the states are not observable (hidden). Two probability density functions are associated to each hidden state: one provides the probability of transition to another state, the other provides the probability that a given symbol is emitted from that state.

According to [18], an HMM $\lambda$ is characterized by the following elements:

- **N**, the number of states in the model.

- **M**, the number of distinct observation symbols per state, i.e. the discrete alphabet size.
- **A**, the state transition probability distribution. In our case is a $NxN$ matrix.
- **B**, the observation symbol probability distribution. In our case is a $NxM$ matrix.
- $\pi$, the initial state distribution. Each element $\pi_i$ is the probability that the initial state is the $i$-th state.

Since *HMMPayl* is an anomaly based system, we can distinguish two different operating phases. In the first phase we train the IDS (that is we estimate its parameters). Then, in the "Detection" phase the IDS can be used to analyze the network traffic. In particular, during the "Training" phase the estimate of the HMM parameters (that is $A, B$ and $\pi$) is calculated with the goal of maximizing the probability assigned by the model to the sequences within the training set. This problem is usually solved iteratively by resorting to the *Baum-Welch* algorithm [27].

During the "Detection" phase the problem is that of estimating the probability of a sequence for the model obtained after training. This problem (usually referred to as "Evaluation") is solved using the *Forward-Backward* procedure [28, 29]. The *Forward-Backward* procedure calculates the probability of the sequence evaluating all the possible sequences of states that can generate the sequence.

### *3.2. Multiple Classifier Systems*

Multiple Classifier Systems (MCS) are widely used in Pattern Recognition applications as they allow to obtain better performance than a single classifier. Reasons why an MCS could perform better than a single classifier have been deeply investigated in the literature, and the effectiveness of MCS for computer security has been also shown [12, 30, 31].

Basically, an MCS exploits the decisions made by an ensemble of classifiers, and combines these decision to obtain a better classification. According to [32], there are at least three reasons for which an ensemble results more accurate and robust of any classifier in the ensemble:

- The **statistical** reason. A learning algorithm can be viewed as a search for the best hypothesis in a space $\boldsymbol{H}$ of hypotheses. In consequence of the finite size of the training set, the learning algorithm will usually

end up with a number of classifiers that achieve the same accuracy on the training data. These classifiers, however, may not produce the same accuracy on unseen data. By constructing an ensemble out all of them, the risk of choosing the wrong classifier can be reduced.

- The **computational** reason. In many cases the optimal training of a classifier is a NP-hard problem: consequently, most learning algorithms usually aim at finding a local optimum of the target function. This optimum usually depends on the starting point. This means that by running the local search from different starting points, and using the obtained classifiers to build an ensemble, a better approximation of the true unknown function can be attained.

- The **representational** reason. In most machine learning applications, the true function for the problem at hand cannot be represented by any of the functions available in $\boldsymbol{H}$. The use of weighted sums of hypotheses drawn from $\boldsymbol{H}$ may allow expanding the space of representable functions.

In this paper we use the MCS paradigm to combine different HMM. A general schema of the proposed HMM ensemble is shown in figure 4. A payload $\mathbf{x}_i$ is submitted to an ensemble $\mathcal{H} = \{HMM_j\}$ of $K$ HMM, each $HMM_j$ produces an output $s_{ij}$, and their outputs are combined into a "new" output $s_i^*$.

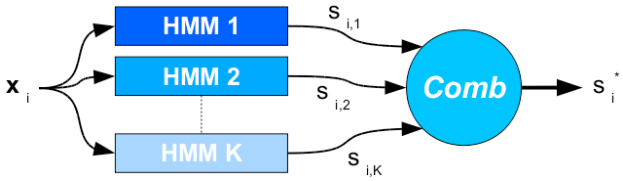


Figure 4: A general schema of a MCS based on HMM.

Different combination strategies for building a MCS have been proposed in the literature. They can be roughly subdivided into two main approaches, namely the *Fusion* approach, and the *Dynamic Selection* approach. In the following, a brief overview of these combination strategies is given.

*3.2.1. Classifier Fusion*

This strategy fuses the outputs of an ensemble of classifiers to produce a single output. A large number of fusion functions is available from the literature, each one with its pros and cons. One of the basic assumptions of

the fusion strategy is that combined classifiers are considered as competitive rather than complementary.

In this paper we will consider the Maximum, the Minimum, the Mean and the Geometric mean rules.

- *the Maximum rule*:

$$s_i^* = \max\{s_{ij}\} \tag{1}$$

- *the Minimum rule*:

$$s_i^* = \min\{s_{ij}\} \tag{2}$$

- *the Mean rule*:

$$s_i^* = \frac{1}{K}\sum_{j=1}^{K} s_{ij} \tag{3}$$

- *the Geometric mean rule*:

$$s_i^* = \left[\prod_{j=1}^{K} s_{ij}\right]^{\frac{1}{K}} \tag{4}$$

These static rules are widely used in Pattern Recognition to combine classifiers because they allow achieving good results in spite of their simplicity. Nevertheless, trained combination rules have been also proposed to better exploit additional knowledge of the domain at hand. As pointed out in [33], combining classifiers using static rules is a suboptimal solution, whereas trained combination rules are asymptotically optimal. Despite this, in this paper we used static rules for two main reasons. One reason is related to the issues involved in building a trained combiner that make its design a non trivial task. The other reason is that static rules are very fast to be computed, and thus the additional computational cost is very small compared to the one typically required by trained combination rules.

*3.2.2. Classifier Selection*

Classifier Selection is based on the assumption that each classifier in a given ensemble exhibits a higher "expertise" than others on a subset of patterns. For each pattern to be classified, the system selects the classifier which is considered to provide the highest accuracy for the pattern at hand. It is easy to see that the main difficulty with this approach is the development of the selection criterion. On the other hand, it can be easily shown that if the selector works properly, very high accuracy can be attained. For this

reason, in this work we will only refer to the so-called "Ideal Score Selector" that provides an upper bound to the performance that could be achieved by the HMM ensemble employed in *HMMPayl*.

The "Ideal Score Selector", proposed in [14], represents the upper bound of the selection strategy. The output of such Ideal Score Selector can be computed as:

$$s_i^* = \begin{cases} \max\{s_{ij}\} & \text{if } x_i \text{ is a } \textit{normal} \text{ pattern} \\ \min\{s_{ij}\} & \text{if } x_i \text{ is an } \textit{attack} \text{ pattern} \end{cases} \quad (5)$$

It can be seen that the selector is "ideal" as the selection function requires the knowledge of the true class the pattern belongs to.

An example of the result attained by the Ideal Score Selector is shown in figure 5, where two classifiers are combined. In particular, for each classifier the distribution of the output values for the two classes is shown. It is easy to see that the distribution of the output values of the Ideal Score Selector allows a better separation between the classes with respect to each of the combined classifiers.

It can be easily seen that the above Ideal Score Selector exhibits a better ROC curve than the ROC curves of each individual classifiers used in the combination, and consequently a larger AUC [14]. Moreover, it has also been shown that the Ideal Score Selector always attains a larger AUC than that obtained by the linear combination, whatever the value of the weights, and the number of classifiers [14].

## 4. IDS Architecture

This section provides the detailed description of the architecture of *HMMPayl*. Since *HMMPayl* is an anomaly based system, it requires a *Training* set made up of normal traffic. The Training phase outputs the Transition and Emission matrices for each HMM included in the IDS. The details on the choice of the initial parameters of the HMM, and on the selection of the training data are reported in section 5.

Here, we will go into details of the *Detection* phase of *HMMPayl*. We have already shown in section 1 that each HTTP payload is processed in three different stages: (a) Feature Extraction, (b) Pattern Analysis and (c) Classification.

In Subsections 4.1, 4.2, and 4.3 we will analyze in detail each processing stage. In addition, we will evaluate the complexity of the algorithm in Subsection 4.4.

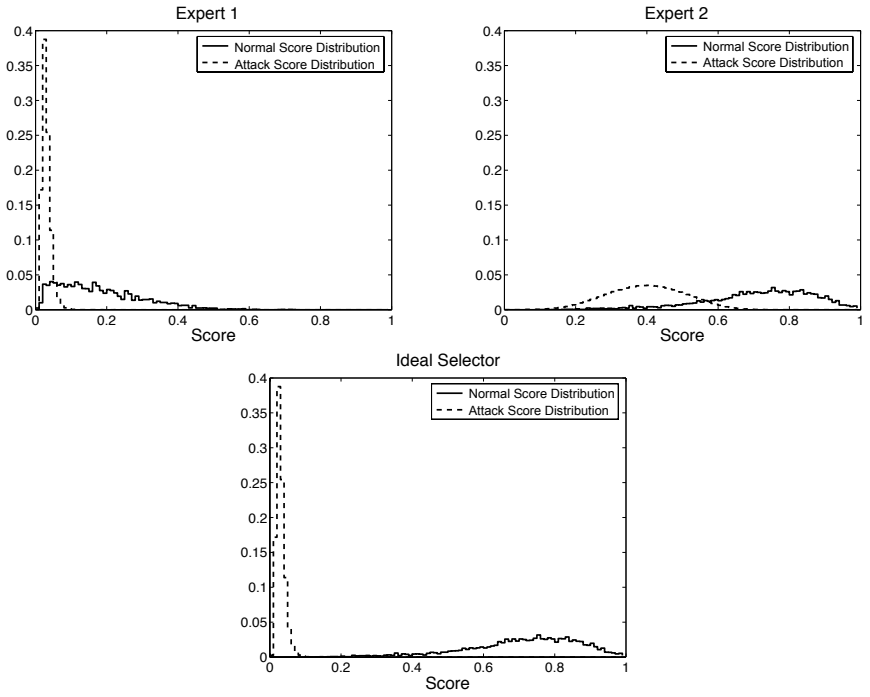


Figure 5: An example of *Ideal Score Selector* with two classifiers on a real dataset. The distributions of the output values resulting from the ideal selection exhibit a larger separability than the original ones.

### *4.1. Feature Extraction*

The most critical task in designing an IDS is that of choosing good features. The main problem in Intrusion Detection is that the "target is moving", i.e., new attacks appear every day, and a number of variants of known attacks are developed, so that attack modeling is a very hard task. Moreover, when the resources to be protected change frequently as in the case of Web applications the problem becomes even more difficult [3]. As a consequence, having good features is crucial because they should not only model the normal traffic, but also allow distinguishing known and possibly never-seen-before attacks. In addition, the features should be chosen in a way that makes difficult for an attacker to craft an attack whose representation in terms of the selected feature is similar to normal traffic. The above considerations have been taken into account when selecting the features to be used in *HMMPayl*.

At a low level, a HTTP payload is not more than a sequence of $L$ bytes, where $L$ is the value of the payload length. As HMM are able to analyze

sequences of data, it would be possible to feed the HMM with the payload as it is. Although it would be a very simple solution, it has two considerable drawbacks. The first one is that the longer the payload, the smaller the probability associated to it. This is a consequence of the fact that the number of products performed by the *Forward-Backward* procedure increases with $L$ (see section 3.1). From a practical point of view, this behavior of HMM increases the risk of having long normal payloads classified as intrusive, while short attack payloads classified as normal.

The second drawback is related to the fact that HMM work better if the length of the processed sequences is equal to (or at least close to) the number of states of the models. This would not be possible if the payload would be processed as it is for several reasons: (a) the length of the payloads may change (b) HMM having hundreds of states are not manageable (the typical length of the HTTP payload is several hundreds of bytes).

These are the reasons why we propose the following approach to represent the payload. A window of width "$n$" slides over the payload byte by byte, and each group of $n$ bytes that falls inside the window is considered as a sequence. Starting from a payload $P$, the sequence extractor produces as output a set $O$ of sequences. Given the length of the payload $L$ and the width of the window $n$, the number of sequences $N$ in $O$ is equal to:

$$N = L - n + 1 \tag{6}$$

In order to better illustrate the features extractor of *HMMPayl*, we will use a toy example. Let us consider the following very short payload made up of 10 bytes, and let us also assume that the bytes can take only three values, namely, 0, 1 and 2.

| 2 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|---|

Let us now consider a sliding window of size $n$ equal to 5. Thus it follows that the number of sequences extracted from the payload is computed as N = 10-5+1 = 6. The set of sequences that are extracted from the toy payload is represented in figure 6.

Thus, a payload $P$ is represented by a set $O$ of $N$ of sequences of length $n$. Each sequence will then be processed by the HMM, and the probability for $P$ of being emitted by the HMM will then be computed as a combination of the outputs for the $N$ sequences in $O$. It is easy to see that payloads with

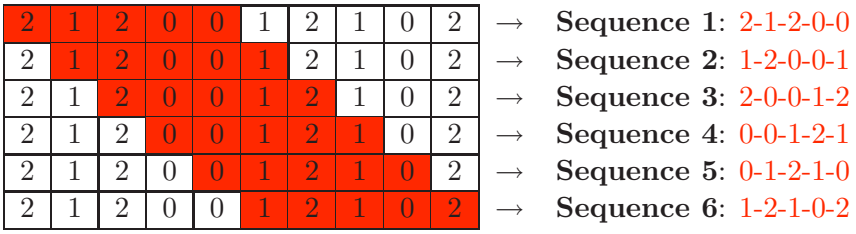


Figure 6: The extraction of sequences from a toy payload

different lengths $L$ will be translated into sets containing a different number of sequences, the size of each sequence being constantly equal to $n$.

Our approach addresses the problem of payload representation, and offers several advantages with respect to other works, namely *PAYL* [9] and *McPAD* [12]. While *PAYL* proposes the use of $n$-gram analysis whose computational cost rapidly increases for values of $n$ greater than 2, the additional computational cost related to the increase of the width "$n$" of the sliding window in our approach is negligible. On the other hand, the $2-\nu-gram$ analysis in *McPAD* is an approximation of the $n$-gram analysis which requires multiple classifiers to be implemented, while the proposed approach models sequences of length $n$ that can be processed serially by a single HMM.

***Sequences Sampling.*** The above example shows that the proposed approach produces some redundancy, as the same byte appears in "$n$" sequences in $O$. We can easily observe that there is an overlap of $n-1$ bytes among a pair of consecutive sequences. In fact, the last $n-1$ bytes of the $t-th$ sequence in the set $O$ are the first $n-1$ of the sequence $t+1$.

As a consequence we expect the following behavior:

- If the distance $i$ among sequences is small (e.g. 1, 2), then the probabilities assigned to sequence $t$ and $t+i$ will be quite similar.
- If the distance $i$ among sequences is large (e.g. $i = n-1$), then the probabilities assigned to sequence $t$ and $t+i$ could not be so close since the overlap among sequences is very small.

This intuition is confirmed by the behavior shown in figure 7. We considered probabilities related to sequences of length $n = 10$, and we varied the distance $i$ from 1 (which means considering consecutive sequences) up to 9 (which means considering just one byte of overlap). Then, for each value of $i$ we calculated the average difference of the probabilities. The figure clearly

shows that the average difference among the probabilities and the standard deviation both increase with the value of $i$. Thus, we can conclude that there is some kind of "redundancy" in the information provided by consecutive sequences.

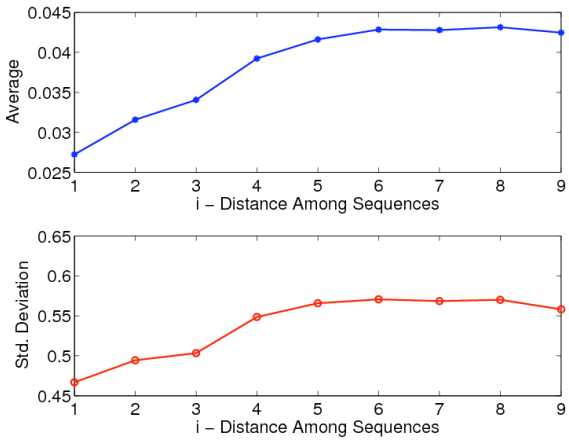


Figure 7: Average and Standard Deviation of the "difference" among the probabilities associated to sequences. The difference is calculated between the probability associated to the sequence extracted at the step t of the "Feature Extraction" process and that assigned by the same HMM to the sequence extracted at the step t+i. The figure refers to the first day of legitimate traffic on the GT dataset.

We exploited this result to reduce the number of sequences analyzed by the HMM. In particular we propose to randomly sample the sequences generated by our system, and use them to classify the payload.

From a practical perspective, the randomization strategy we propose has several outcomes. First of all, it speeds up the IDS as much as we reduce the sampling ratio. Moreover, randomization might be also a defense strategy against attempts of evasion [34]. As the subset of analyzed sequences is chosen randomly for each payload, an attacker is unaware of the portions of the payload used to classify it. Consequently, the evasion of the IDS is a harder task. While this is an interesting issue, in this paper we are not able to show any quantitative evaluation of the difficulty of evading the IDS. On the other hand, reported experiments in sections 6.4 and 7 show the performance of the system when 20%, 40%, 60% and 80% of sequences are randomly chosen from the whole set of sequences.

*4.2. Pattern Analysis*

The input for this step is the set $O$ of $N$ sequences extracted from a payload $P$. As the pattern to be classified by *HMMPayl* is the payload $P$, we will now illustrate how we calculate the probability for $P$ of being emitted by the HMM.

First of all, one HMM is trained on sets of sequences extracted from normal payloads according to the technique previously described. Then, at detection time, for each payload $P$, the HMM calculates the probability of emitting each one of the sequences inside $O$:

$$p_j = P(o_j|\lambda),\ j = 1, ..., N \tag{7}$$

Thus, for each payload $P$, the HMM produces a set of $N$ probabilities associated to the $N$ sequences in $O$. A simple solution to obtain an overall probability for $P$ is to calculate the *arithmetic mean* of the output probabilities:

$$P(O|\lambda) = \frac{1}{N}\sum_{j=1}^{N} p_j = \frac{1}{N}\sum_{j=1}^{N} P(o_j|\lambda) \tag{8}$$

as the arithmetic mean provides an efficient and unbiased estimate.

The scheme of *HMMPayl* reported in figure 3 shows that $K$ different HMM are used in parallel during the Pattern Analysis step. We have provided in section 3.2.1 several reasons that make an ensemble of classifiers more suitable than a single one. Consequently, we decided to make the payload being analyzed in parallel by an ensemble of HMM. As each HMM processes all the payload, all the HMM in the ensemble have the same number of states, and are trained on the same training set. The difference among the HMM in the ensemble is in the random initialization of the $A$ and $B$ matrices during the training phase, which produces different final matrices. Summing up, each HMM receives as input the whole set of sequences $O$, and produces as output a set of $N$ probabilities. The set of output probabilities from each HMM is the input for the "Sequences Probabilities Fusion" block which calculates the arithmetic mean of the $N$ probabilities produced by each HMM. At the end of this step we obtain a vector of $K$ probabilities assigned to the payload by the $K$ HMM.

*4.3. Classification*

At this level, the problem is that of combining the outputs of several classifiers. We have already discussed in section 3.2 the reasons why a system based on multiple classifiers should be preferred to a system based on a

single classifier. We would just remind here that both the accuracy and the difficulty of evading the IDS benefit from such an approach [12, 16, 31].

Among all of the available techniques for combining an ensemble of classifiers we decided to use the static combination rules described in section 3.2.1, i.e., the minimum, the maximum, the mean, and the geometric mean rules [30]. The output of the MCS block is thus the probability that the payload $P$ belongs to the normal traffic. A payload $P$ is then classified as normal or attack if the probability assigned by the system is respectively above or below a predefined threshold. The choice of the threshold depends on the accepted trade-off between detection and false positive rates.

### *4.4. Complexity Evaluation*

Now we will provide an analysis of the computational complexity of the *HMMPayl* algorithm. Since the training of *HMMPayl* is performed off-line, here we evaluate the complexity of the detection phase only.

Given a payload P of length $L$ and a value of $n$ for the length of the sequences extracted from the payload, the extraction of sequences from the payload can be accomplished in $O(L-n+1)$. Then, the number of sequences extracted increases linearly with the length of the payload. Consider that typical values for the length of the HTTP payload are in the order of several hundreds of bytes. Thus a number of several hundreds of sequences has to be extracted from the payload. On the contrary, in the case of *PAYL* a number of features must be calculated which exponentially increases with the value of $n$ (it is $256^n$) [9]. In the case of *McPAD* the number of features that must be calculated for each payload received by the IDS is 65536 ($256^2$) [12].

Once sequences have been extracted, the likelihood of each sequence (that is the probability that a sequence is emitted by an HMM) has to be calculated using the *Forward-Backward procedure* [18]. The *Forward-Backward procedure* requires the calculation of $n$ Forward variables plus the calculation of $n$ Backward variables. If $s$ is the number of states of the HMM and T is the length of the sequence to be processed, according to Rabiner each one of the Forward and Backward variables required to calculate the likelihood can be calculated in $O(s^2 \cdot T)$ [18]. Since in the case of *HMMPayl* $s = T = n$, the computational cost becomes $O(n^3)$. Given that $O(n^3)$ is the number of computations required for each variable, since for each sequence we have to calculate $2 \cdot n$ variables, the number of computations required for each sequence is $O(n^4)$. Since the number of sequences extracted from each payload is $N = L - n + 1$ for each classifier within the ensemble a number of operations $O(N \cdot n^4)$ has to be performed.

## 5. Experimental Setup

This section is organized as follows. Subsections 5.1, 5.2 and 5.3 provide some guidelines on how to setup the system, and the choices we made to perform the reported experiments.

Subsection 5.4 describes the datasets used to validate the proposed model. In particular we used different datasets of normal traffic, and attacks. Finally, Subsection 5.5 details the measures we used to evaluate the performance of *HMMPayl*.

### *5.1. The width n of the sliding window*

From the previous section, it should be clear that the larger the width of the window, the more accurate the IDS. The experimental results reported in the following section aim at confirming this hypothesis. We varied the value of $n$ from 2 to 10 to evaluate how the accuracy of *HMMPayl* increases with the length of the sequences. It is also worth noting that further increase in the window size does not provide significant gain in performance, as distant bytes are loosely related.

### *5.2. Hidden Markov Models Parameters*

In section 3.1 it has been shown that the performance of HMM are affected by the choice of its parameters, namely (a) the number of hidden states; (b) the initialization of emission and transition matrices. A brief description on how to set these parameters follows.

*Number of States.* The estimation of the most appropriate number of hidden states of an HMM for a given application is more art than science. Anyway, there are several heuristics that allow building effective HMM. In particular, we observed that a good choice of the number of hidden states is related to the length of the sequences to be processed [16]. Thus, we have chosen to build the HMM with a number of states exactly equal to the length $n$ of the sequences. Therefore, in our experiments the number of states varies in the range from 2 to 10 according to the value of $n$. It is worth to notice that all the HMM in the ensemble have the same number of states, the only difference being the initialization of the matrices.

*Initialization of Matrices.* The behavior of an HMM resulting from the training procedure depends on the training data as well as on the initial values of the matrices A and B (see section 3.1). Among all the strategies available to initialize the matrices A and B, we decided for the random initialization [18]. In fact, other strategies different from random initialization

generally try to take into account the structure of sequences to be modeled. Since in *HMMPayl* this structure is arbitrary (as it depends from the value of $n$) these strategies are not suitable for our purposes. Furthermore, random initialization is a common practice in Pattern Recognition when the selection of parameters can't be driven by data.

### *5.3. Number of HMM*

After some preliminary experiments we found that a number of HMM $K = 5$ was a good compromise between system complexity and its ability being accurate, hard to evade, and fast enough.

### *5.4. Datasets*

In this section we describe the characteristics of the datasets we used in our experiments. *HMMPayl* has been deeply tested on three different datasets of normal traffic, and on five datasets containing different types of attacks.

#### *5.4.1. Normal Traffic*

In our experiments three datasets of normal traffic have been employed, one of them containing simulated traffic, while the other two contain real traffic collected at academic institutions.

The dataset containing simulated traffic consists of the HTTP requests extracted from the first week of the DARPA'99 dataset [35]. Altogether it is composed of five entire days of simulated normal traffic to and from an air force base. Although the DARPA dataset is outdated and has been criticized for the way it was generated, to the best of our knowledge it was the only public dataset of network traffic at the time we performed the experiments [8, 36]. Very recently some researchers have made publicly available a dataset of network traffic collected during a network warfare competition (ITOC dataset) [37]. Anyway, it appeared too late to give us the opportunity of including some results in this paper.

The other two datasets containing traces of real traffic have been collected at two academical institutions. One dataset is made up of HTTP requests towards the website of the College of Computing at the Georgia Institute of Technology (GT), USA. The other dataset consists of requests toward the website of the Department of Electrical and Electronic Engineering (DIEE) at the University of Cagliari, Italy. They consist respectively of seven and six days of traffic. It is important to remark that both the GT and the DIEE datasets are completely unlabeled. Anyway, it is reasonable to assume that the vast majority of the traffic is made up of legitimate

Table 1: Number of packets and size (MB) of traces of normal traffic

| | DARPA | | DIEE | | GT | |
|---|---|---|---|---|---|---|
| **Day** | **Size (MB)** | **Packets** | **Size (MB)** | **Packets** | **Size (MB)** | **Packets** |
| 1 | 19 | 161,602 | 7.2 | 10,200 | 131 | 307,929 |
| 2 | 23 | 196,605 | 7.4 | 10,200 | 72 | 171,750 |
| 3 | 23 | 189,362 | 6,6 | 10,200 | 124 | 289,649 |
| 4 | 30 | 268,250 | 6 | 10,200 | 110 | 263,498 |
| 5 | 18 | 150,847 | 6.4 | 10,200 | 79 | 195,192 |
| 6 | – | – | 6.7 | 10,763 | 78 | 184,572 |
| 7 | – | – | – | – | 127 | 296,425 |

HTTP requests, and even if we cannot exclude the presence of attacks, they would represent a very negligible fraction with respect to the normal traffic. For this reason, we consider the possible fraction of attacks in the dataset as noise. In fact, both networks are protected by firewalls and IDSs, and in case of persistent intrusion attempts, there is usually evidence that an attack is occurring. As any evidence has not been reported in the period in which we collected the traffic, we assume that the level of noise in both datasets is negligible. Therefore, in the following we consider the GT and DIEE datasets as "clean" from known attacks for the purpose of measuring the false positive rate.

The experiments have been carried out in the same way on all the three datasets, both for training and testing. A $k$-fold cross validation has been realized, using in rotation one day of traffic for training and all the remaining days for testing purposes. Details about the number of packets and the size (in MB) of each trace are provided in table 1.

*5.4.2. Attack Datasets*

We experimented *HMMPayl* with several datasets made up of the most frequently observed attacks against Web applications. In particular, we used a publicly available dataset of HTTP attacks provided by the authors of [38]. We further created polymorphic variants of these attacks by using the polymorphic engine CLET [39]. Moreover, we created a dataset of attacks that exploited vulnerabilities of the Web applications executed by the Web server at DIEE. With the exception for this last set of attacks, all the other attacks used in the experiments are the same that have been used in [12]. For the sake of organizing the results, we grouped the attacks in a number of datasets, each one containing attacks of the same category. In the following, we will briefly provide a detailed description of each dataset.

- **Generic Attacks**. This dataset includes all the HTTP attacks pro-

vided by the authors of [38] plus a shell-code attack that exploits the vulnerability MS03-022 of the Windows Media Service (WMS), which was used in [40]. 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. The other attacks cause Information Leakage, and Denial of Service (DoS).

- **Shell-code Attacks**. This dataset contains the 11 shell-code attacks contained in 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 worm, like Code-Red, for example, use shell-code attacks to propagate.

- **CLET Attacks**. This dataset contains 144 polymorphic attacks generated using the polymorphic engine CLET [39]. We selected 8 among the 11 *Shell-code Attacks*, and created polymorphic versions of each attack using the payload statistics computed on each distinct day of traffic from the DARPA, GT and DIEE datasets for training CLET's polymorphic engine. The total number of attacks generated by CLET is equal to 144.

- **XSS-SQL Attacks** This dataset of attacks has been also used in [16]. It consists of a set of custom attacks that exploit vulnerabilities of the Web applications executed by the Web Server at DIEE, the same Web server used to collect the DIEE dataset of normal traffic. This set of attacks has been created as follows. We selected a set of attacks published on `www.milw0rm.com`, and used these attacks as the basis on which building a set of attacks exploiting specific vulnerabilities of the applications running on the Web server. The resulting dataset consists of 19 SQL Injection attacks and 19 XSS attacks. References for these attacks are reported in table 2, where details are given in terms of the identifying number of the exploit, and the paper where the vulnerability is described.

### *5.5. Performance Evaluation*

#### *5.5.1. ROC Curves*

In order to validate the classification performance of our detector, we use the Receiver Operating Characteristic (ROC) curve analysis, and the related Area Under the ROC Curve (AUC). The ROC curve provides a way

Table 2: References for attacks inside XSS-SQL Dataset. Attacks are taken from www.milw0rm.com. For each attack the number identifying the exploit and that of the paper where the vulnerability is described are provided.

| **Attack Type** | **Exploit N.** | **Paper N.** |
|---|---|---|
| **SQL Injection** | 6512, 6510, 6502, 6490, 6469, 6467, 6465, 6449, 6336, 3490, 5507 | 16, 174, 202, 215 |
| **XSS** | 2776, 2881, 2987, 3405, 3490, 4681, 4989, 6332 | 162, 173, 192 |

to visually represent the trade-off between false positive and detection rates by varying the detection threshold [41]. While the measure of accuracy depends on a specific value of the detection threshold, 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 (i.e., the higher the AUC, the easier to distinguish attacks from normal traffic) [42].

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 of no interest to evaluate the performance of an Intrusion Detection System when a high number of false alarms are generated, we focused our analysis on a small range of the false positive rate. In particular, we computed 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 the performance of the Intrusion Detection System for values of the false positive rate greater than 10%. In order to obtain a performance value in the range $[0, 1]$, we normalized the "partial" AUC computed in $[0, 0.1]$ by dividing it by 0.1. In the following, we will denote the performance measured by the "partial" AUC as $AUC_p$.

Finally, as *HMMPayl* does not reconstruct HTTP sessions, we consider a per packet detection rate. This means that the detection rate reported in our experiments is related to the fraction of attack packets detected by the IDS. This is a pessimistic estimate of the real detection rate that would be calculated in terms of the fraction of attacks detected. In real cases, an attack is considered to be detected if at least one of its packets is detected.

## 6. Experimental Results

This section shows the performance of *HMMPayl*, and it is organized in five parts. Section 6.1 shows the performance of *HMMPayl* on both Shell-code and CLET attacks, while sections 6.2 and 6.3 show the ability

of the system in detecting Generic and XSS-SQL attacks, respectively. The behavior of the system when a randomly sampled subset of sequences are used for classification is discussed in section 6.4. Section 6.5 reports the upper bound in performance attained by the Ideal Score Selector. In Section 6.6 we evaluate the "sensitivity" of *HMMPayl* to the presence of anomalous bytes within the payload. Section 6.7 provides a comparison of *HMMPayl*, *McPAD*, and *Spectrogram* in terms of Detection Rate at fixed values of the False Positive Rate. Finally, Section 6.8 gives an evaluation of the benefits provided by the architecture based on Multiple Classifiers.

### *6.1. Shell-code and CLET dataset*

The results attained on the Shell-code and the CLET datasets are shown together, as the attacks in the CLET dataset have been obtained by modifying Shell-code attacks with a polymorphic engine so that the $n$-gram statistics match those of the normal traffic. Thus, all of the considered attacks aim at injecting executable code into the target machine, where CLET attacks should be more difficult to detect as they have been crafted to be *similar* to normal traffic. Table 3 shows the average and the standard deviation of the partial AUC ($AUC_p$) calculated for the whole range of the considered length of sequences (namely, from 2 to 10), and by considering all the combining rules illustrated in section 4. Reported results show that, for this category of attacks, the length of sequences does not affect the accuracy significantly, as the values of the standard deviation are quite small. The reported values of $AUC_p$ are quite high, thus pointing out that the proposed system is effective in detecting not only Shell-code attacks, but also their polymorphic variants that have been crafted to evade the detection system.

Table 3: Resume of the values of $AUC_p$ of *HMMPayl* for Shell-code and CLET attacks

| | **Minimum** | | **Maximum** | | **Mean** | | **GeoMean** | |
|---|---|---|---|---|---|---|---|---|
| **DARPA** | | | | | | | | |
| **Attack** | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ |
| **Shell-code** | 0.99909 | 0.0019376 | 0.91937 | 0.065238 | 0.97187 | 0.041408 | 0.99967 | 0.00030095 |
| **CLET** | 0.9984 | 0.002354 | 0.93551 | 0.05324 | 0.97553 | 0.039972 | 0.99935 | 0.00059503 |
| **DIEE** | | | | | | | | |
| **Attack** | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ |
| **Shell-code** | 0.9966 | 0.0024 | 0.8812 | 0.0857 | 0.9319 | 0.0843 | 0.9945 | 0.0051 |
| **CLET** | 0.9974 | 0.0051 | 0.9128 | 0.0800 | 0.9495 | 0.0768 | 0.9989 | 0.0022 |
| **GT** | | | | | | | | |
| **Attack** | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ | $\overline{AUC_p}$ | $\sigma$ |
| **Shell-code** | 0.9887 | 0.0035 | 0.8547 | 0.0937 | 0.9153 | 0.0659 | 0.9865 | 0.0043 |
| **CLET** | 0.9906 | 0.0087 | 0.8887 | 0.0825 | 0.9434 | 0.0596 | 0.9915 | 0.0094 |

Such a good result can be easily explained if we inspect figure 8 where the

distribution of 2-grams for the normal traffic and for the Shell-code attacks is shown. The 2-grams of normal traffic are concentrated in the down-left corner of the graph as the normal traffic contains only ASCII characters. On the contrary, the bytes of Shell-code attacks are distributed evenly in the range 0-255. As the training set is made up of normal traffic, the vast majority of training sequences is made up of ASCII characters, and the probability associated by HMM to non-ASCII symbols is close to zero. This makes quite easy for *HMMPayl* to recognize Shell-code attacks.

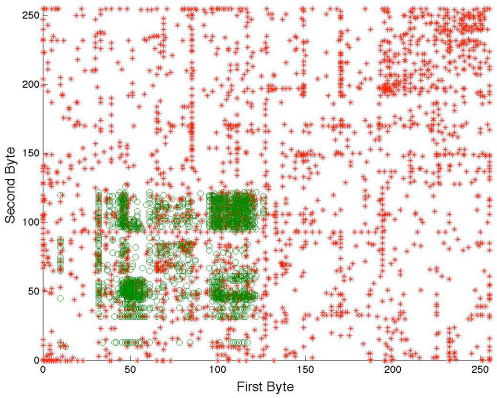


Figure 8: Distribution of bytes from normal traffic (green) and from Shell-code attacks (red).

Furthermore, the comparison of different combining rules shows that the minimum rule allows attaining the best performance. This result is not surprising, and it is also supported by a theoretical background [43]. In fact, it is known that the minimum rule is conceptually equivalent to the logic operator *AND* [44]. This means that combining the outputs of several classifiers using a minimum rule sounds like finding a complete consensus among all the classifiers in the ensemble.

Finally, in figure 9 we present an example of the ROC curves obtained on the GT dataset obtained when a value $n = 10$ is chosen for the length of the sequences. The figure shows the curves for the different days of legitimate traffic, and in addition an average curve obtained averaging the false positive and the detection rate on the different days. This plot contains a remarkable result. In fact even when a very low false positive rate is considered (approximately $10^{-4}$) *HMMPayl* achieves a very high detection rate (higher than 0.85).

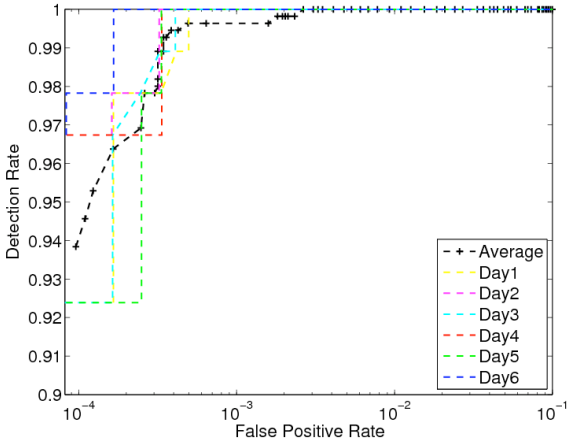


Figure 9: ROC curves for Shellcode Attacks on the DIEE dataset. The n parameter has been set to 10. The "Average" curve has been obtained by averaging the detection rate on the six days of network traffic. The minimum rule has been used to combine classifiers.

### *6.2. Generic Dataset*

The "Generic" dataset includes different attack types, including Shell-code attacks. While Shell-code attacks can be detected quite easily, other attacks included in this dataset (e.g. DoS, URL decoding error, etc.) are more difficult to be detected by IDS such as *PAYL* or *McPAD*, as it has been pointed out in section 1. In particular, let us recall that the statistics of a payload containing a Denial of Service attack (see for example figure 2) do not significantly deviate from those of normal traffic. This implies that a detailed model of the payload is necessary to achieve a high detection rate. *HMMPayl* can provide such a detailed model as it allows attaining higher detection rates than those of similar approaches in the literature.

Experimental results are presented in figure 10 where *HMMPayl* is compared to *McPAD* on both the DARPA and the GT datasets. *PAYL* is not considered in this comparison because it has been already shown that *McPAD* outperforms *PAYL* when these attacks are considered [12]. *McPAD* performance is averaged on the number of clusters, and is obtained considering the combination rule that performed the better. The considered rules are the product of probabilities on the DARPA dataset, and the minimum on the GT and DIEE dataset respectively (see [12]). Reported results for *HMMPayl* are related to the minimum rule.

The graph clearly shows that the $AUC_p$ attained by *HMMPayl* increases with the value of $n$. This means that the larger the value of $n$, the better the

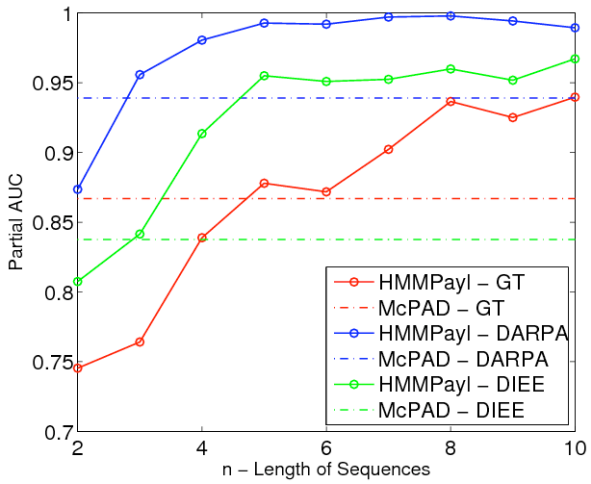


Figure 10: $AUC_p$ values for the Generic Attacks Dataset. The $AUC_p$ increases with the length $n$ of sequences extracted from the payload. *HMMPayl* classifiers are combined using the minimum rule.

structure of the payload is inferred by the IDS. In particular, a value of $n = 3$ is sufficient for *HMMPayl* to outperform *McPAD* on the DARPA and DIEE datasets, whereas on the GT dataset the same behavior is attained with a value of $n$ greater than or equal to 5. A possible explanation for this might be the fact that the traffic in the GT dataset is more complex with respect to that in the DARPA and DIEE datasets. As a consequence, a larger value of $n$ is needed to obtain an accurate model of the higher variability of the payload observed in real traffic.

To conclude this section, let us have a look on figure 11 where we present complete ROC curves. For false positives rates higher than $10^{-3}$ *HMMPayl* attains a detection rate always higher than 0.85 which is a remarkable result especially if we recall the characteristics of attacks within this dataset. As the false positive rate reduces, the percentage of detected attacks obviously decreases but still remains higher than 70% for a false positive rate of $10^{-4}$.

### *6.3. XSS and SQL-Injection Attacks*

When *HMMPayl* is tested on Cross Site Scripting (XSS) and SQL-Injection attacks, the behavior is similar to that exhibited by *HMMPayl* when tested on "Generic Attacks" . This similarity is not surprising since the structure of the payload of attacks belonging to the XSS-SQL dataset is quite similar to that of normal traffic, as well as some of the attacks inside the "Generic Attacks" dataset (see section 6.2). Accordingly, the value

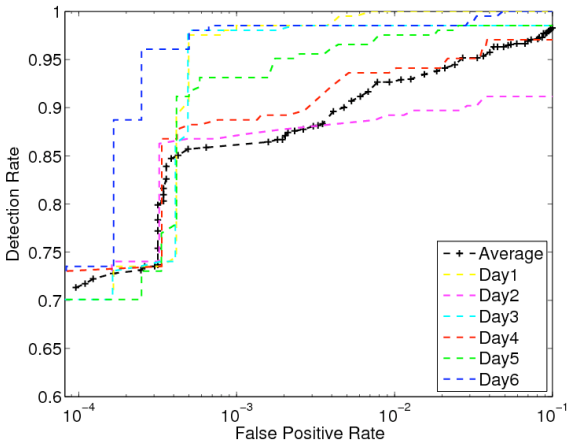


Figure 11: ROC curves for Generic Attacks on the DIEE dataset. The n parameter has been set to 10. The "Average" curve has been obtained by averaging the detection rate on the six days of network traffic. The minimum rule has been used to combine classifiers.

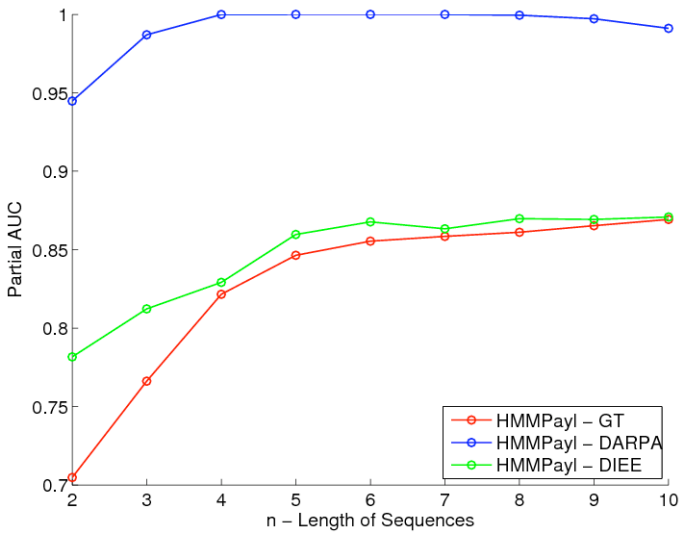


Figure 12: Values of $AUC_p$ for the XSS and SQL Injection Attacks. The $AUC_p$ increases with the length $n$ of sequences extracted from the payload. *HMMPayl* classifiers are combined using the minimum rule.

of $AUC_p$ increases with the length $n$ of the sequences extracted from the payload (figure 12).

It is worth pointing out two aspects arising from the analysis of the

results reported in figure 12. One comment is related to the behavior on the GT dataset, which is definitely the most difficult to model. In spite of this difficulty, the $AUC_p$ provided by *HMMPayl* is larger than 0.85, and a value of $n = 6$ is sufficient to attain this result.

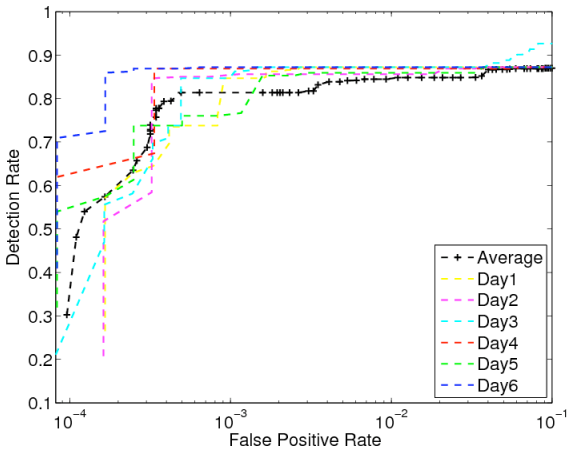


Figure 13: ROC curves for XSS-SQL Attacks on the DIEE dataset. The n parameter has been set to 10. The "Average" curve has been obtained by averaging the detection rate on the six days of network traffic. The minimum rule has been used to combine classifiers.

Figure 13 shows the complete ROC curves. The detection rate is definitely good for values of the false positive rate higher than $10^{-3}$. Below this value the average detection rate decreases even if on certain days of traffic (e.g. "Day 6") it remains quite good. These results are reasonable if we consider how Cross-Site Scripting and SQL-Injection attacks work. In fact they basically exploit flaws in the validation of the input provided to Web applications. Then, they affect only a certain portion of the payload.

To conclude this section, we provide a comparison of *HMMPayl* with *McPAD* [12], *Spectrogram*[15], and *HMM-Web* [16]. We averaged the performance of each IDS on the six days of traffic from the DIEE dataset.

For the purpose of this comparison we set:

- A value of 8 for the $\nu$ parameter in *McPAD*. A 2-$\nu$-gram analysis with $\nu = 8$ approximates a 10-gram analysis. We set to 0.01 the desired false positive rate, the number of clusters to $k = 160$, and we used the maximum rule for combining classifiers.

- A value of 10 for the gram-size in *Spectrogram*, and a number of 5 Markov-chains within the mixture.

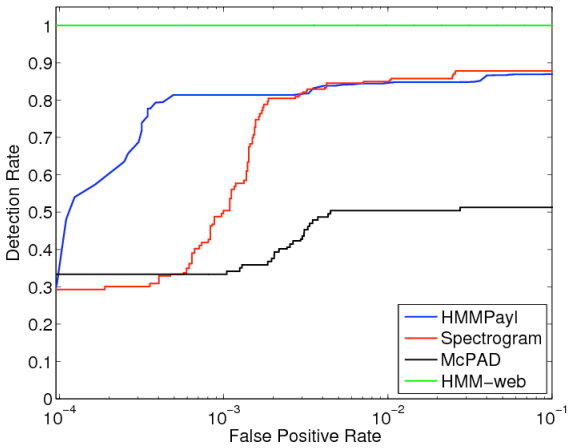


Figure 14: Comparison of *HMMPayl*, *Spectrogram*, *McPAD*, and *HMM-Web* on the XSS-SQL Injection attack dataset. ROC curves are averaged on the six days of the DIEE dataset.

- A number of 5 HMM within each ensemble used by *HMM-Web*.
- A length $n = 10$ for the sequences analyzed by *HMMPayl*, and a number of 5 HMM in the ensemble. We used the minimum rule to combine HMM.

Results are shown in figure 14. As expected XSS and SQL-Injection attacks are particularly hard for *McPAD*. In fact this IDS is able to detect at most 50% of them producing a very high false positive rate. *HMMPayl* performs significantly better since it is able to detect a high percentage of attacks (more than 80%) even for low false positive rates. In fact, at 0.01 % false positives rate the IDS is still able to detect more than 50% of the attacks. With respect to *Spectrogram* we can observe that *HMMPayl* exhibits almost the same performance for false positives rates higher than 0.1%. In spite of this fact, as the false positive rate reduces *HMMPayl* performs significantly better. For instance, at 80% of detection rate the percentage of false alarms generated by *HMMPayl* is approximately one order of magnitude smaller than that generated by *Spectrogram*. More in general, the detection rate achieved by *HMMPayl* decreases more slowly as the false positive rate reduces. *HMM-Web* provided the highest performance since it has been able to detect all the attacks even for very low false positives.

These results can be easily explained if we remind the architecture of *HMM-Web*, *McPAD*, and *Spectrogram* that we described in section 2. In

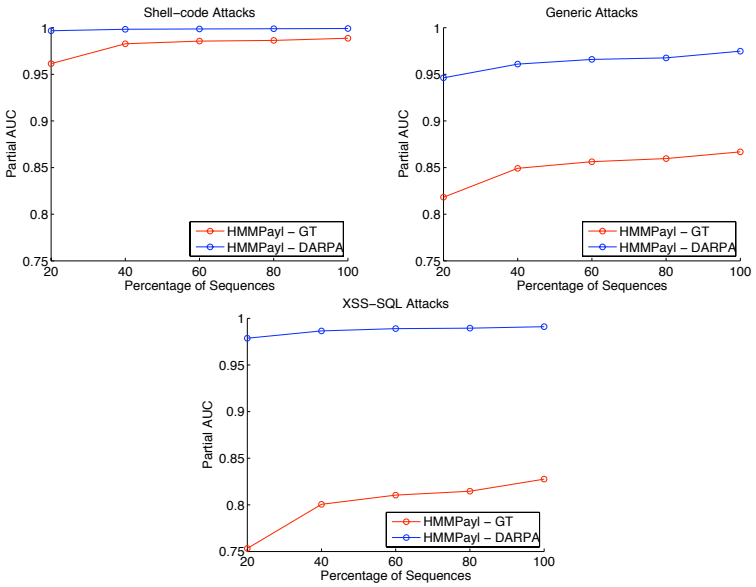


Figure 15: Performance of *HMMPayl* in terms of $AUC_p$ when a subset of sequences is randomly chosen. The sampling varies between 20% and 100%. Classifiers are combined using the minimum rule.

particular *HMMPayl*, *McPAD*, and *Spectrogram* rely on a single model that analyses all the traffic toward the Web server, whereas *HMM-Web* has a different set of models for each application on the Web server.

### *6.4. Sequences Sampling*

In section 4 we proposed a randomization strategy to exploit the redundancy in the sequences used to classify the payload. We expect a decrease in performance related to the amount of reduction of the number of sequences used in the classification phase. However, it is of interest to quantify the amount of decrease, in order to measure the trade-off between computational complexity and accuracy. To this end, we tested the performance of *HMMPayl* on the DARPA and GT datasets, as they are more difficult to model than the DIEE dataset. Attacks from the Shell-code, XSS-SQL and Generic datasets have been considered, while the ensemble of HMM has been combined by the minimum rule. Experimental results are reported in figure 15.

As we expected, the larger the percentage of sequences randomly sampled, the larger the $AUC_p$, the maximum $AUC_p$ being achieved when the whole packet is considered (100%, i.e., no sampling is performed). Anyway

it is worth to remark that *HMMPayl* attains good values of $AUC_p$ even if a very small percentage of sequences is considered (e.g. 20-30%). For example, at a sampling rate of 20% on the GT dataset, the corresponding reduction in $AUC_p$ is approximately 3% for Shell-code attacks, 6% for Generic attacks and 9% for attacks into XSS-SQL. If the 40% of sequences are considered, the loss reduces to 0.6% , 2% and 3% respectively. The performance loss on the DARPA dataset is always smaller than that observed on the GT dataset since the normal traffic in the DARPA dataset is more simple to be modeled. We can also observe that the loss is larger for those attacks (such as XSS) whose detection is strictly related to the amount of information that can be extracted from the payload. It is straightforward to see that these results make sense if they are compared with the reduction in the per-packet processing time. By considering just the 20% of sequences, the cost of payload processing is approximately reduced by a factor of 4. A more detailed discussion on the computational cost will be provided in section 7.

Another point which is worth to remark is that the random sampling of sequences might be also useful to make harder for an attacker the task of evading the IDS. In principle, if we random sample the sequences, the attacker does not know at which part of the payload we are looking at to classify it. Then, he would be able to reproduce the whole payload structure in order to evade the IDS. On the other hand, a deterministic sampling of the sequences based on the characteristics of the attacks may allow reducing the performance loss. However, the measure of the trade-off between detection accuracy, difficulty of evasion, and computational cost is out of the scope of this paper.

### *6.5. Ideal Score Selector*

In this section we present the results attained by applying the "Ideal Score Selector" described in section 3.2.2 compared with the results obtained using the Minimum Rule. Reported results on classifier selection have been carried out on the DARPA and the GT datasets, and aim to show the gain in performance that could be attained by a careful design of the classifier fusion module. In particular, as the performance attained by *HMMPayl* in the case of Shell-code and CLET attacks are very high (see section 6.1), the experiments in figure 16 are restricted to the Generic and XSS-SQL attacks datasets. If the "right" HMM is selected, according to the rule showed in section 3.2.2, it is possible to increase the $AUC_p$ up to 100%. In other words, the use of the ensemble of HMM provide complementary information that can be further exploited to increase the performance. It is straightforward to see that if more complex combination rules are used

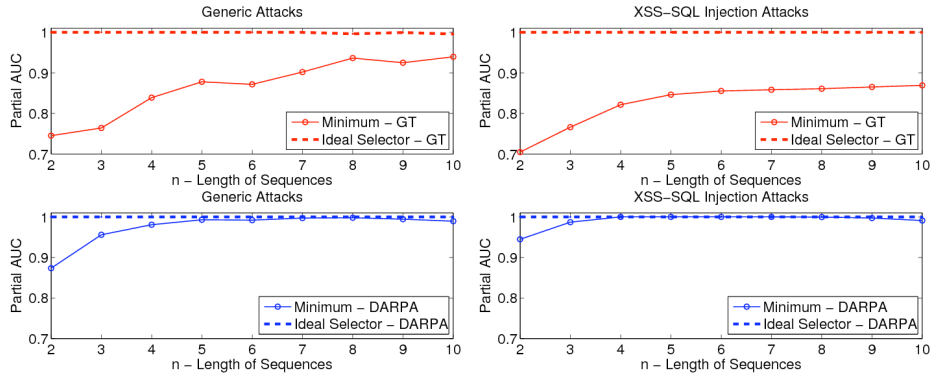


Figure 16: Comparison of the $AUC_p$ attained by the Ideal Score Selector with that attained by the Minimum Rule. GT and DARPA datasets of normal traffic, and Generic and XSS-SQL datasets of attacks are considered.

instead of the Minimum rule, the performance can be closer to those of the Ideal Score Selector. However, the task of deploying the most suitable combination rule could be as hard as the solution of the detection problem itself, as it would require resorting to "trained" fusion rules [33].

### *6.6. Sensitivity to Anomalies*

In this section we evaluate the sensitivity of *HMMPayl* to the presence of anomalies within the payload. For the way it has been performed, this evaluation can be considered also as a rough estimate of the robustness of *HMMPayl* against attempts of evasion. For the purpose of this assessment we created an ensemble of payloads containing artificial anomalies starting from a set of legitimate payloads. Artificial anomalies have been employed both for training and testing purposes in Intrusion Detection as well as in Spam Detection and Biometrics [45, 46, 47].

We applied the following procedure to generate artificial anomalies. We randomly selected from each payload a number of bytes according to a fixed percentage of the size of the payload. Then, we replaced the value of each one of the selected bytes by another byte value randomly chosen. More in detail, we experimented with two different substitution strategies. In the first case, the substitution has been performed replacing the value of the original byte with a value in the range $[0, 127]$. This to simulate anomalies that contain ASCII characters only. In the second case, the values of substituting bytes have been randomly chosen within the range $[0, 255]$.

We set a value for the threshold such that all the packets without modifications are correctly classified as legitimate. For this value of the threshold

*HMMPayl* is able to detect approximately the 85% of the attacks within the Shell-code dataset and the 70% of those within the Generic dataset. After, we generated a different set of payloads containing anomalies simply varying the percentage of modified bytes. We experimented with a percentage of modifications from 1% up to 60% with intermediate steps of 1% within the range 1-10%, and with intermediate values 15, 20, 30, 40 and 50% in the range 10-60%. We limited to 60% the percentage of modified bytes since results related to higher percentages of modified bytes are not of interest for this evaluation. In fact we are particularly interested in evaluating how the IDS behaves when a small percentage of bytes is anomalous.

We expect the IDS achieving a high detection rate against those payloads for which a large percentage of byte has been modified. In addition, we also expect that payloads containing only ASCII characters will be more hardly detected by the IDS, since they closely resemble legitimate payloads.

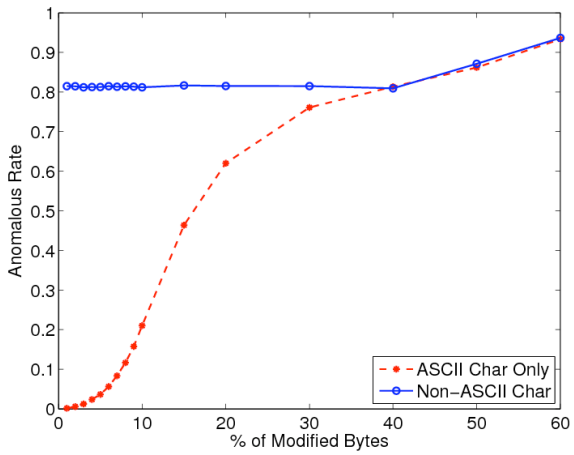


Figure 17: Sensitivity of *HMMPayl* to the presence of anomalies within the payload. Experiments have been performed on the first day of the DIEE dataset. Sequences length has been set to 6.

Experimental results are shown in figure 17. As expected, payloads containing non ASCII characters are easily labeled as anomalous by the IDS. The red curve is definitely more interesting. We can observe that to be classified as legitimate with a probability higher than 0.9 a payload must contain at least an 80% bytes that resembles the structure of a legitimate payload.

Even if strategies of evasion more and more sophisticated are continuously developed (e.g. [48]) we are quite confident that in a real case it would

be quite difficult for an attacker to produce attacks that are so similar to legitimate payload.

### *6.7. Performance evaluation in terms of Detection Rate at fixed values of the False Positive Rate*

The measure of performance used so far provided an overall evaluation of the IDS in the range of false alarm rates from 0% to 10%. While this is a useful measure as its value does not depend on a particular threshold value, it is also useful to inspect the behavior of *HMMPayl* for some particular values of the threshold. A typical performance measure is the evaluation of the detection rate by setting the threshold so that the false positive rate is equal to 0.01 and 0.001, that is, 1% and 0.1% of false alarms, respectively. These values are shown in table 4 for the DARPA and GT datasets of normal traffic, and for all the attack datasets.

Table 4: *HMMPayl* detection rate at false positive rate = 0.01 and 0.001. Classifiers are combined using the minimum rule. Performance refers to sequence of length $n = 8$.

| **Dataset** | **False Positive** | | Detection Rate | | | |
|---|---|---|---|---|---|---|
| | **Desired** | **Real** | **XSS-SQL** | **Generic** | **Shell-code** | **CLET** |
| **DARPA** | 0.01 | 0.0103 | 1 | 0.984 | 0.997 | 0.996 |
| | 0.001 | 0.0013 | 0.996 | 0.941 | 0.989 | 0.984 |
| **GT** | 0.01 | 0.01 | 0.86 | 0.897 | 1 | 1 |
| | 0.001 | 0.001 | 0.805 | 0.779 | 0.978 | 0.991 |

These results are consistent with those presented in the previous sections, thus confirming the validity of the proposed approach. In fact, when a percentage of false alarms equals to 1% is allowed, *HMMPayl* detects all the attacks in the Shell-code and CLET datasets. At the same percentage of false alarms, approximately the 90% and 94% of attacks in the Generic dataset is detected on the GT and DARPA datasets, respectively. When a percentage of false alarms of 0.1% is allowed, the detection rate still remains very high with respect to attacks in the Shell-code and CLET datasets. As far as attacks in the Generic and XSS-SQL datasets are concerned, *HMMPayl* guarantees approximately a detection rate of 80% at least. These results clearly point out the effectiveness of the proposed model.

Table 5 shows the comparison of *HMMPayl* with *McPAD* and *Spectrogram* on the DIEE dataset. The setup of *HMMPayl*, *McPAD*, and *Spectrogram* is the same considered in section 6.3.

Generally, at the same false positive rate *HMMPayl* attains a detection rate higher with respect to *McPAD*. A particularly remarkable result is that obtained on the XSS-SQL attack datasets. As expected *HMMPayl* performs

Table 5: *HMMPayl*, *McPAD*, and *Spectrogram* detection rate on the DIEE dataset at false positive rate 0.01, 0.001, and 0.0001.

| **Desired FP** | **IDS** | **Real FP** | Detection Rate | | |
|---|---|---|---|---|---|
| | | | **XSS-SQL** | **Generic** | **Shell-code** |
| **0.01** | ***HMMPayl*** | $(7.28 \cdot 10^{-4})$ | 0.844 | 0.926 | 1 |
| | ***McPAD*** | $( 5.5 \cdot 10^{-3})$ | 0.504 | 0.824 | 0.998 |
| | ***Spectrogram*** | $( 1.3 \cdot 10^{-5})$ | 0.849 | - | - |
| **0.001** | ***HMMPayl*** | $(3.57 \cdot 10^{-4})$ | 0.813 | 0.858 | 0.996 |
| | ***McPAD*** | $( 1.69 \cdot 10^{-7})$ | 0.333 | 0.720 | 0.967 |
| | ***Spectrogram*** | $( 4.69 \cdot 10^{-6})$ | 0.495 | - | - |
| **0.0001** | ***HMMPayl*** | $(3.9 \cdot 10^{-6})$ | 0.302 | 0.713 | 0.938 |
| | ***McPAD*** | $( 3.45 \cdot 10^{-7})$ | 0.333 | 0.629 | 0.962 |
| | ***Spectrogram*** | $( 5.62 \cdot 10^{-6})$ | 0.19 | - | - |

definitely better than *McPAD* against these attacks. For instance, at 0.1% of false positives *HMMPayl* is able to detect more than 81% of attacks whereas *McPAD* only 33%. If we consider a 0.01% of false positives and the XSS-SQL attack dataset, *McPAD* slightly outperforms *HMMPayl*. Nevertheless it must be considered that since the detection rate achieved by both the IDS is around 30%, this is not an operating point at which both the IDS can be used in practice.

As we mentioned in section 2 *Spectrogram* is able to detect only attacks that exploit flaws in the validation of the input provided to the Web application. Then we restricted the comparison to the XSS-SQL attack dataset. Results in table 5 show that at 1% of false positives rate *HMMPayl* and *Spectrogram* achieve approximately the same detection rate (the difference is around 0.5%). Nevertheless as the amount of false positive reduces *HMMPayl* performs significantly better than *Spectrogram*. In particular at 0.1% of false positives rate, *Spectrogram* detects approximately 50% of attacks whereas *HMMPayl* exhibits a detection rate higher than 80%.

### *6.8. Benefits of the MCS approach*

In this section we briefly provide experimental evidence of the benefits provided by the employment of multiple classifiers. Let us first observe figure 18 where the partial AUC has been calculated considering a number of classifiers from 1 to 5. Classifiers are combined using the minimum rule. All the possible combinations of 2, 3, and 4 classifiers have been considered and the average $AUC_p$ has been calculated. The figure refers to the first day of the GT dataset and to a value $n = 6$ for the width of the window. Results obtained on different days of traffic and for different values of $n$ are similar.

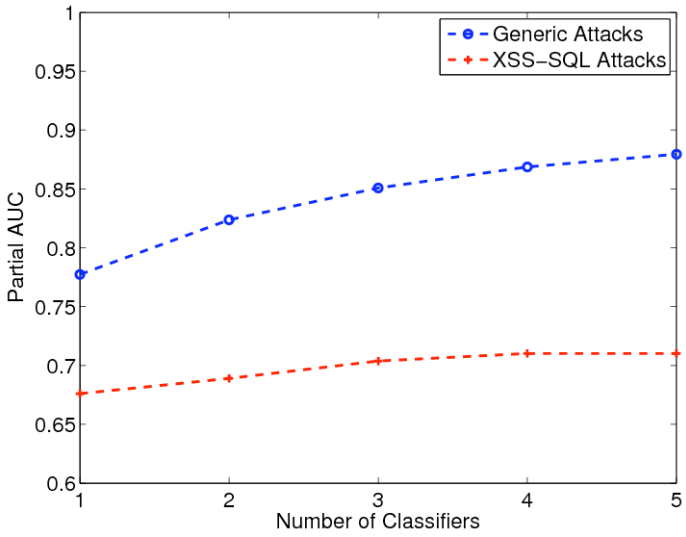


Figure 18: Increase of the $AUC_p$ with the number of classifiers. The $AUC_p$ is the average of those obtained considering all the possible combinations of 2, 3, and 4 classifiers. Classifiers are combined using the minimum rule.

It is important to observe that the $AUC_p$ increases as the number of classifiers combined. We limited to five the number of classifiers in the ensemble since from some preliminary experiments we observed that five was a good trade-off between accuracy and computational cost. In particular the figure clearly shows that the increase of the $AUC_p$ obtained adding a classifier to the ensemble becomes smaller as the size of the ensemble increases.

In addition, let us consider table 6 where the $AUC_p$ achieved by single classifiers is compared to that achieved using all the five HMM and the minimum rule.

Table 6: Value of the $AUC_p$ for individual classifiers. The MCS includes all of the five HMM combined with the minimum rule.

| | **HMM1** | **HMM2** | **HMM3** | **HMM4** | **HMM5** | **MCS** |
|---|---|---|---|---|---|---|
| **Generic** | 0.8036 | 0.8163 | 0.8564 | 0.8973 | 0.7944 | 0.918 |
| **XSS-SQL** | 0.8559 | 0.8584 | 0.6155 | 0.8221 | 0.6717 | 0.8546 |

With respect to **XSS-SQL** attacks we can observe that HMM3 and HMM5 perform very poorly. In spite of their presence within the ensemble, the $AUC_p$ achieved by the ensemble is only of a 0.44% smaller of that achieved by the best classifier. This result is particularly interesting if we

consider that it is not trivial to establish which classifier will provide the best performance on different test sets. This is clearly shown in table 6 where HMM3 performs very bad on XSS-SQL attacks while it is the second best (after HMM4) on the Generic dataset. Further on the Generic attack dataset, the $AUC_p$ obtained with the MCS is higher than that achieved by the best classifier in the ensemble.

## 7. Analysis of the Computational Cost

In this section, we provide a brief discussion on the computational cost of *HMMPayl*. At the moment, *HMMPayl* is just a set of scripts that implement the algorithm. This means that we do not implemented an optimized software tool which is able to work on line, reading packets from the network interface and assigning a probability to them. Consequently, the computational cost reported here is just an average per packet cost, that has been calculated during the testing phase. No evaluation of the computational cost has been made during the training phase, as it can be performed off-line, and thus it is not required that the IDS is able to keep up with the network speed.
Table 7 reports the characteristics of the machines on which we ran the experiments.

Table 7: Workstation Specifics

| **CPU** | **Memory** | **O.S.** |
|---|---|---|
| 2 Dual Core AMD Opteron(tm) Processor 280 | 8 GB | Debian 4.0 |

The scripts that implement the *HMMPayl* algorithm are based on two libraries:

- LibPcap [49], which is a network programming library used to extract bytes from the HTTP payload.
- GHMM [50], which is a library that implements Hidden Markov Models. In particular, we used the Python wrapper of GHMM.

Tables 8, 9 and 10 show the performance of *PAYL*, *McPAD*, and *HMMPayl* respectively in terms of average processing time per packet. It can be observed that *HMMPayl* is definitely slower than *PAYL*, even in the cases when *HMMPayl* process just a small subset of the sequences extracted for each packet. On the other hand, performance of *McPAD* and *HMMPayl* are quite similar and higher than those of *PAYL*. We are aware that the

Table 8: *HMMPayl*'s average processing time per payload. The value between brackets represents the number of payloads in the test dataset. The sampling ratio indicates the percentage of sequences sampled from each payload with the randomization strategy. *HMMPayl* uses m=5 HMM.

| Dataset | Sampling Ratio | | | | |
|---|---|---|---|---|---|
| | 20 % | 40 % | 60 % | 80 % | 100% |
| **DARPA (137,997)** | 7.48 ms | 12.71 ms | 17.70 ms | 22.96 ms | 27.51 ms |
| **GT (1,068429)** | 8.37 ms | 14.18 ms | 20.72 ms | 27.59 ms | 32.66 ms |

Table 9: *McPAD*'s average processing time per payload. The value between brackets represents the number of payloads in the test dataset. m is the number of classifiers used by the IDS. For the definition of cluster please refer to [12].

| Dataset | Number of Clusters | FP = 0.001 m=3 | FP = 0.001 m=11 | FP = 0.01 m=3 | FP = 0.01 m=11 |
|---|---|---|---|---|---|
| **DARPA (137,997)** | k=10 | 3.07 ms | 10.96 ms | 3.16 ms | 11.04 ms |
| | k=40 | 3.04 ms | 11.02 ms | 3.11 ms | 11.31 ms |
| | k=160 | 3.13 ms | 11.92 ms | 3.81 ms | 13.39 ms |
| **GT (1,068429)** | k=10 | 4.28 ms | 16.23 ms | 4.94 ms | 17.53 ms |
| | k=40 | 4.16 ms | 15.92 ms | 6.14 ms | 21.93 ms |
| | k=160 | 4.95 ms | 17.11 ms | 10.49 ms | 38.45 ms |

Table 10: *PAYL*'s average processing time per payload. The value between brackets represents the number of payloads in the test dataset.

| Dataset | Processing Time |
|---|---|
| **DARPA (137,997)** | 0.039 ms |
| **GT (1,068429)** | 0.032 ms |

reported processing time of *HMMPayl* is not suitable for an IDS with real time constraints. Anyway, we can provide several reasons that motivate the slowness of our implementation:

**Proof of concept** The implementations of both *HMMPayl* and GHMM are proof of concept, aimed at evaluating the performance in terms of detection and false alarm rates.

**Python** Both *HMMPayl* and GHMM are written in Python, which is a fast interpreted language but obviously it is not as fast as compiled languages, such as C or C++.

These performance could be easily improved in several ways. Among them, we can consider:

- Implementing the IDS in C or $C++$. LibPcap, which is probably the most widely used for network programming, is also written in C [51].

- Multi-thread processing. As in *HMMPayl* the payload analysis is made by a set of classifiers, a parallel calculation which exploits recent multi-core architectures would allow to speed-up the IDS.

For these reasons, we are very confident on the fact that a good implementation of the algorithm would be able to significantly improve the performance in terms of processing time.

It is also worth remarking the benefits provided by the random sampling in terms of computational cost. In section 6.4 we have already discussed how the random sampling strategy affects the accuracy of *HMMPayl*. It has been shown that even if a small percentage (20%) of the sequences extracted from the packet is considered, the $AUC_p$ still remains very high, as being approximately the 97% of that obtained considering the whole packet. At the same time, as it can be observed from table 8 the IDS becomes 4 times faster. For this reason, our opinion is that sampling the sequences produced for each payload can be seen as a way to speed-up the IDS.
To conclude, we provide some informations on the amount of memory used by the algorithm: the memory employed by the process was 2-3% of the on-board memory (200-300 MB) which is a negligible amount respect to that available on modern machines. The optimization of the code will reduce certainly also the amount of memory employed.

## 8. Conclusions and Future Works

In this paper we proposed an IDS designed to detect attacks against Web applications through the analysis of the HTTP payload by HMM. With this work we provided for several innovative contribution.

First of all we proposed a new approach for extracting features which exploits the power of HMM in modeling sequences of data. Reported experiments clearly show that this approach provide a statistical model of the payload which is particularly accurate, as it allows detecting attacks effectively, while producing a low rate of false alarms. *HMMPayl* has been thoroughly tested on three different datasets of normal traffic, and against four different dataset of attacks. In particular, we showed that *HMMPayl* was able to outperform other solutions proposed in the literature. In particular, *HMMPayl* is effective against those attacks such as Cross Site Scripting and SQL-Injection, whose payload statistic is no significantly different from that of normal traffic. These attacks are particularly hard to be detected, as the performance of IDS such as *PAYL* and *McPAD* clearly show.

In addition, we also showed that the high computational cost of *HMMPayl* can be significantly reduced by randomly sampling a small percentage

of the sequences extracted from the payload, without significantly affecting the overall performance in terms of detection and false alarm rates.

Moreover, as *HMMPayl* relies on the Multiple Classifier System paradigm, we tested the performance attained by the Ideal Score Selector as a measure of the maximum gain in performance that could be attained by exploiting the complementarity of the HMM. Experimental results show that the accuracy can be improved with an accurate design of the fusion stage.

It is clear that, despite the good results attained in our experiments, the algorithm implemented by *HMMPayl* could be further improved. First of all, *HMMPayl* does not take into account the length of the payload. As different lengths of the payload produce significantly different statistics, clustering the payloads by length, and using a different model for each cluster, would improve the overall accuracy. The second improvement is related to the random sampling strategy, as the whole sequence set could be randomly split among all the classifiers in the ensemble. In such a way all the information inside the payload would be used, where a single HMM is asked to process a smaller number of sequences. Finally, the third improvement is related to the use of trained combination rules instead of a static rule to combine the *HMM*.

## Acknowledgments

This research was sponsored by the RAS (Autonomous Region of Sardinia) through a grant financed with the "Sardinia PO FSE 2007-2013" funds, and provided according to the L.R. 7/2007 for the "Promotion of the Scientific Research and of the Technological Innovation in Sardinia". We authorize the RAS to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the RAS.

The authors would like to thank Prof. Salvatore J. Stolfo who gently provided them a copy of *Spectrogram* for the purpose of their experiments.

## References

[1] Internet Security Systems - IBM-ISS, X-force 2009 trend and risk report, Tech. rep., IBM Global Technology Services (2010).

[2] Breach Security Inc. - ModSecurity: Open Source Web Application Firewall, http://www.modsecurity.org (November 2009).

[3] F. Maggi, W. K. Robertson, C. Krügel, G. Vigna, Protecting a moving target: Addressing web application concept drift, in: E. Kirda, S. Jha, D. Balzarotti (Eds.), RAID, Vol. 5758 of Lecture Notes in Computer Science, Springer, 2009, pp. 21–40.

[4] Breach Security Inc. - WebDefend (November 2009).
URL http://www.breach.com/products/webdefend.html

[5] Citrix Systems Inc. - Netscaler Application Firewall, http://www.citrix.com (November 2009).
URL http://www.citrix.com/English/PS2/products/product.asp?contentID=25636

[6] F5 Networks Inc. - BIG-IP Application Security Manager (November 2009).
URL http://www.f5.com/products/big-ip/product-modules/application-security-manager.html

[7] C. Krügel, T. Toth, E. Kirda, Service specific anomaly detection for network intrusion detection, in: SAC '02: Proceedings of the 2002 ACM symposium on Applied computing, ACM, New York, NY, USA, 2002, pp. 201–208.

[8] M. V. Mahoney, Network traffic anomaly detection based on packet bytes, in: SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, ACM, New York, NY, USA, 2003, pp. 346–350.

[9] K. Wang, S. J. Stolfo, Anomalous payload-based network intrusion detection, in: E. Jonsson, A. Valdes, M. Almgren (Eds.), RAID, Vol. 3224 of Lecture Notes in Computer Science, Springer, 2004, pp. 203–222.

[10] K. Wang, G. F. Cretu, S. J. Stolfo, Anomalous payload-based worm detection and signature generation, in: A. Valdes, D. Zamboni (Eds.), RAID, Vol. 3858 of Lecture Notes in Computer Science, Springer, 2005, pp. 227–246.

[11] K. Wang, J. J. Parekh, S. J. Stolfo, Anagram: A content anomaly detector resistant to mimicry attack, in: D. Zamboni, C. Krügel (Eds.), RAID, Vol. 4219 of Lecture Notes in Computer Science, Springer, 2006, pp. 226–248.

[12] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, W. Lee, Mcpad: A multiple classifier system for accurate payload-based anomaly detection,

Computer Networks 53 (6) (2009) 864 – 881, special Issue on Traffic Classification and Its Applications to Modern Networks.

[13] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1 (1999).

[14] R. Tronci, G. Giacinto, F. Roli, Dynamic score selection for fusion of multiple biometric matchers, in: R. Cucchiara (Ed.), ICIAP, IEEE Computer Society, 2007, pp. 15–22.

[15] Y. Song, A. D. Keromytis, S. J. Stolfo, Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic, in: NDSS, The Internet Society, 2009.

[16] I. Corona, D. Ariu, G. Giacinto, HMM-Web: A framework for the detection of attacks against web applications, in: Communications, 2009. ICC '09. IEEE International Conference on, 2009, pp. 1–6.

[17] M. Damashek, Gauging similarity with n-grams: Language-independent categorization of text, Science 267 (5199) (1995) 843–848.

[18] L. Rabiner, A tutorial on hidden markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (1989) 257–286.

[19] S. Günter, H. Bunke, Optimizing the number of states, training iterations and gaussians in an hmm-based handwritten word recognizer, in: Proceedings of the Seventh International Conference on Document Analysis and Recognition, IEEE Computer Society, 2003, p. 472.

[20] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological sequence analysis, Cambridge University Press, 2006.

[21] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: alternative data models, in: Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on, 1999, pp. 133–145.

[22] S.-B. Cho, S.-J. Han, Two sophisticated techniques to improve HMM-based intrusion detection systems, in: G. Vigna, E. Jonsson, C. Krügel (Eds.), RAID, Vol. 2820 of Lecture Notes in Computer Science, Springer, 2003, pp. 207–219.

[23] D. Gao, M. Reiter, D. Song, Beyond output voting: Detecting compromised replicas using HMM-based behavioral distance, Dependable and Secure Computing, IEEE Transactions on 6 (2) (2009) 96–110.

[24] C. Kruegel, G. Vigna, Anomaly detection of web-based attacks, in: CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, ACM, New York, NY, USA, 2003, pp. 251–261.

[25] C. Y. Suen, n-gram statistics for natural language understanding and text processing, Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-1 (2) (1979) 164–172.

[26] P. Fogla, 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, ACM, New York, NY, USA, 2006, pp. 59–68.

[27] L. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains, The Annals of Mathematical Statistics 41 (1) (1970) 164–171.

[28] L. Baum, J. Egon, An inequality with applications to statistical estimation for probabilistic function of a markov process and to a model for ecology, Bulletin American Metereology Society 73 (1967) 360–363.

[29] L. Baum, G. Sell, Growth functions for transformations on manifolds, Pacific Journal of Mathematics 27 (2) (1968) 211–227.

[30] L. Kuncheva, Combining Pattern Classifiers, Wiley, 2004.

[31] I. Corona, G. Giacinto, C. Mazzariello, F. Roli, C. Sansone, Information fusion for computer security: State of the art and open issues, Information Fusion 10 (2009) 274–284.

[32] T. G. Dietterich, Ensemble methods in machine learning, in: J. Kittler, F. Roli (Eds.), Multiple Classifier Systems, Vol. 1857 of Lecture Notes in Computer Science, Springer, 2000, pp. 1–15.

[33] R. Duin, The combining classifier: to train or not to train?, in: Pattern Recognition, 2002. Proceedings. 16th International Conference on, Vol. 2, 2002, pp. 765–770 vol.2.

[34] B. Biggio, G. Fumera, F. Roli, Adversarial pattern classification using multiple classifiers and randomisation, in: N. da Vitoria Lobo, T. Kasparis, F. Roli, J. T.-Y. Kwok, M. Georgiopoulos, G. C. Anagnostopoulos, M. Loog (Eds.), SSPR/SPR, Vol. 5342 of Lecture Notes in Computer Science, Springer, 2008, pp. 500–509.

[35] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, K. Das, The 1999 darpa off-line intrusion detection evaluation, Computer Networks 34 (4) (2000) 579 – 595, recent Advances in Intrusion Detection Systems.

[36] 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) (2000) 262–294.

[37] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, J. Adams, C. Morrell, G. Conti, Toward instrumenting network warfare competitions to generate labeled datasets, in: Security's Workshop on Cyber Security Experimentation and Test (CSET), USENIX, 2009.

[38] K. L. Ingham, H. Inoue, Comparing anomaly detection techniques for HTTP, in: C. Krügel, R. Lippmann, A. Clark (Eds.), RAID, Vol. 4637 of Lecture Notes in Computer Science, Springer, 2007, pp. 42–62.

[39] T. Detristan, T. Ulenspiegel, Y. Malcom, M. Underduk, Polymorphic shellcode engine using spectrum analysis, Phrack 0x0b (0x3d).

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

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

[42] C. Cortes, M. Mohri, Confidence intervals for the area under the roc curve, in: Advances in Neural Information Processing Systems (NIPS 2004), Vol. 17, MIT Press, 2005.

[43] B. Biggio, G. Fumera, F. Roli, Multiple classifier systems for adversarial classification tasks, in: J. A. Benediktsson, J. Kittler, F. Roli (Eds.), MCS, Vol. 5519 of Lecture Notes in Computer Science, Springer, 2009, pp. 132–141.

[44] L. Kuncheva, Fuzzy Classifier Design, Vol. 49 of Studies in Fuzziness and Soft Computing, Springer-Verlag, 2000.

[45] W. Fan, M. Miller, S. Stolfo, W. Lee, P. Chan, Using artificial anomalies to detect unknown and known network intrusions, Knowledge and Information Systems 6 (5) (2004) 507–527.

[46] A. Kolcz, C. H. Teo, Feature weighting for improved classifier robustness, in: Sixth Conference on Email and Anti-Spam (CEAS), Mountain View, CA, USA, 2009.

[47] R. N. Rodrigues, L. L. Ling, V. Govindaraju, Robustness of multimodal biometric fusion methods against spoof attacks, Journal of Visual Language and Computing 20 (3) (2009) 169–179.

[48] J. Mason, S. Small, F. Monrose, G. MacManus, English shellcode, in: CCS '09: Proceedings of the 16th ACM conference on Computer and communications security, ACM, New York, NY, USA, 2009, pp. 524–533.

[49] Libpcap: Network programming library, http://www.tcpdump.org.

[50] Ghmm: General hidden markov model library, http://ghmm.org/.
URL `http://ghmm.org/`

[51] L. MartinGarcia, Programming with libpcap - sniffing the network from our own application, Hakin9 Magazine (February 2008).
URL `http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf`