Machine learning-assisted virtual patching of web applications

Gustavo Betarte*†, Eduardo Giménez†, Rodrigo Martínez*† and Álvaro Pardo‡
*Instituto de Computación, Facultad de Ingeniería Universidad de la República, Uruguay
Email: [gustun,rodmart]@fing.edu.uy
†Tilsor SA, Uruguay
Email: [gbetarte,egimenez,rmartinez]@tilsor.com.uy
‡Departamento de Ingeniería Eléctrica, Facultad de Ingeniería y Tecnologías
Universidad Católica del Uruguay, Uruguay
Email: apardo@ucu.edu.uy

***Abstract*—Web applications are permanently being exposed to attacks that exploit their vulnerabilities. In this work we investigate the application of machine learning techniques to leverage Web Application Firewall (WAF), a technology that is used to detect and prevent attacks. We propose a combined approach of machine learning models, based on one-class classification and n-gram analysis, to enhance the detection and accuracy capabilities of MODSECURITY, an open source and widely used WAF.**

The results are promising and outperform MODSECURITY when configured with the OWASP Core Rule Set, the baseline configuration setting of a widely deployed, rule-based WAF technology. The proposed solution, combining both approaches, allow us to deploy a WAF when no training data for the application is available (using one-class classification), and an improved one using n-grams when training data is available.

***Index Terms*—Web Application Firewalls, Machine Learning, Anomaly Detection, One-class Classification, n-grams**

I. INTRODUCTION

A web application is a piece of software, based on a client-server architecture, that embodies a coordinated set of functions. The information flowing between the client, which runs on the user's web browser, and the application server is transmitted using the HTTP protocol.

By its very nature web applications are designed to be exposed, therefore they are available to any individual, or artifact, with capabilities to access the Internet. Thus, web applications are a primary target for any attacker who wants to unauthorizedly access the information they handle or to place baits (e.g., fake URLs to his own site) to lure honest users. It is quite usual for the code of a web application to contain vulnerabilities like the ones listed and described in the OWASP TOP 10 [1]. The work reported in this paper focuses on preventing the abuse of web applications.

A Web Application Firewall (WAF) is a piece of software that intercepts and inspects all the traffic between the web server and its clients, searching for attacks inside the HTTP packet contents. Once recognized, the suspicious packets may be processed in a different, secure way, for instance being logged, suppressed or derived to a honeypot application.

MODSECURITY [2] is an open source, widely used WAF enabling real-time web application monitoring, logging and access control. The actions MODSECURITY undertakes are driven by rules that specify, by means of regular expressions, the contents of the HTTP packets to be spotted. MODSECURITY offers a default set of rules, known as the OWASP Core Rule Set (OWASP CRS) [3], for tackling the most usual vulnerabilities included in [1]. However, an approach only based on rules also has some drawbacks: rules are static and rigid by nature, so the OWASP CRS usually produces a rather high rate of false positives, which in some cases may be close to 40% [4]. Rule tuning is a time consuming and error prone task that has to be manually performed for each specific web application. In traditional networks firewalls and IDS, the approach based on rules has been successfully complemented with other machine learning-based tools, anomaly detection and other statistical approaches which provide higher levels of flexibility and adaptability. Those approaches take advantage of sample data, from which the normal behavior of the web application can be learned, in order to spot suspicious situations which fall out of this nominal use (anomalies), and which could correspond to on-going attacks. Our final objective is to improve MODSECURITY with such anomaly detection techniques.

The design of a statistically-based WAF may be built applying the knowledge that a security expert has on both the application itself and the current state of the art on attack techniques. Alternatively, a description of the expected normal behavior of the web application may be built from legitimate users' input. In the first approach the WAF is trained to recognize payloads that are instances of known attacks and any input which cannot be recognized as an attack is deemed to be a valid input. The second approach adopts the symmetric point of view: the WAF learns the expected inputs of the web application and any other bias from the expected values is deemed as a potential attack. As we shall show in what follows both approaches can be composed to recognize a broader class of web application attacks.

This research was partially funded by ICT4V (https://www.ict4v.org). The results reported here are the result of work carried out in the context of WAFINTL, a project focused on WAF technology.

Organization of the paper: The structure of the rest of the paper is as follows: Section II provides some background and describes the approach we have taken to enhance MODSECURITY virtual patching capabilities with machine-learning techniques. Section III presents the two complementary learning models that we use to ground a statistical WAF. Then, in Section IV we describe and discuss the outcomes of the experiments we have performed. Section V reviews related work. Further work and conclusions are presented in Section VI.

II. BACKGROUND AND PROBLEM STATEMENT

A. Web application security and virtual patching

Usually, successful attacks result from exploiting vulnerabilities present in the web application. The inspection and fixing of the application's code, however, might not always be feasible. Critical applications, for instance, can not be off-line until the bug has been fixed. Moreover, it might well be the case that the source code of the application is neither accessible nor longer available or patchable. We are interested in the use of security *virtual patching* as a remediation technique. In particular, we seek to enhance the functioning of the widely deployed WAF called MODSECURITY. This virtual patcher can be used to protect any web application and its working relies on standardized and public protocols such as HTTP.

Traditional network firewalls are designed to perform packet inspection at the network level. Web applications, however, exchange messages in higher layers, mostly HTTP, that traditional firewalls are not designed to understand. MODSECURITY has been specifically designed to protect web applications with the ability to detect and identify patterns that could result in an attack against the application. It is able to intercept a broad class of attacks, including, among others, XSS (cross-site scripting), any form of code injection and path traversal. MODSECURITY has two modes of operation: detection and prevention. In the first mode, records are generated for each detected potential attack and used to monitor specific rules. Normally, this mode is used in what is called learning phase. The second mode is when the WAF is really useful: by correctly configuring rules and directives it is able to block potential malicious Web traffic. The core of MODSECURITY implements a rule engine which can be set to execute in each transaction of the application.

The use of MODSECURITY provides a first line of defense against attacks and allows the detection of security problems using a rule-based system. In other words, it is capable of enforcing a negative security policy by blacklisting requests that are filtered by those rules. This frequently results in the generation of false alarms and requires a continuous update of those rules. Failure to do so may result in limited or no protection against new (zero-day) attacks and the generation of too many false positives, which must be adjusted by adding exceptions to the rules for each application. These adjustments are strictly necessary in order to set to work the firewall, otherwise the number of false positives generated can be so high that basically results in a denial of service. However, the management of rule configurations for MODSECURITY is a complex and error-prone task. In particular, the writing and maintenance of these configurations requires mastering a low level language with an intricate syntax in which the modeling of complex attacks involves several rules possibly residing in multiple configuration files.

B. Learning attack detection

MODSECURITY is usually configured to work using a negative security policy because defining the rules that describe normal behavior of the application to protect is an almost impossible task in real life. The problem with this operational mode is the high amount of false positives generated and the amount of work needed during the learning phase. The main objective of the research reported here is to enhance MODSECURITY operation by using machine learning techniques to adapt its behavior to that of the application that is protecting. Depending on the operational context of the target application one may consider alternative learning scenarios.

The problem of protecting a web application can be addressed using two different automated learning approaches: as a *classification* problem and as an *anomaly detection* problem. In the first approach protection of the application amounts to classify a new given input using a model that has been constructed from a previous training phase based on supervised learning and using a collection of inputs, each one labeled either as a valid input or as some kind of attack from an attack taxonomy. However, the ideal situation of having available a labeled dataset of application requests that represent valid and attack behavior of a specific application is not always possible. The second approach focuses on characterizing which are the expected (valid) input values for the web application considering any anomalous input supplied to the application as a potential attack. Protecting the application therefore is equivalent to outlier detection from a training dataset of valid inputs.

The anomaly detection approach has several advantages. First, as any behavior different from the usual one is considered problematic, it can potentially make it possible to detect zero-days attacks. Additionally, training the WAF only requires a collection of HTTP requests produced by friendly users and no need for labelling the HTTP requests. However, as noticed in [5], this approach cannot provide an answer to all possible attack scenarios. This is because some of the web application input data may be random data by its very nature. Consider for instance a web application parameter carrying a user password. It can be made of any combination of symbols, and in principle all combination should be equally likely. In such cases there is no hope to find a strong language signature for the field, nor to reject a chosen password on the basis that it is not frequently used. Furthermore, for security reasons, it would not be advisable for the WAF to store information about the distribution of such sensible parameters. In those cases, where there is no a priori expected behavior, it is necessary to adopt a symmetric approach, and search for attack signatures in the field according to the current state of the art. This amounts to search for carefully selected tokens which have

been extracted from a labeled training dataset and representing the knowledge of the security expert on the current attack vectors. This technique deems the HTTP request as valid by default when it cannot be recognized as an attack according to the previous training.

C. Improving MODSECURITY detection capabilities

We have been experimenting with different learning techniques to enhance the detection and accuracy capabilities of MODSECURITY. We have pursued a combined approach that we now proceed to explain.

First, we have experimented with a mechanism that integrates one-class classification and MODSECURITY configured using the OWASP CRS rules out of the box. This basically combines two experts with the objective of classifying a request. When both experts agree (both say valid or both say attack) then the result is straightforward. In the case the one-class approach classifies a request as an attack, given that the OWASP CRS rules have the know-how on attacks, we prioritize its answer (so if MODSECURITY classifies the request as valid then is valid). This integration decision also allows us to have a well known mechanism to train our WAF in the case we found a false positive. Basically we need to tell MODSECURITY rules that this request is not an attack. This training mechanism is exactly the same as the one used with the OWASP CRS nowadays. Finally, in the case that one-class classifies the request as valid and MODSECURITY as an attack we prioritize one-class since OWASP CRS is known to have high false positive rates. This approach has shown to adapt quite well to the scenario where there is not available a specific training dataset for the application. We train the one-class classifier using several datasets and the resulting classification model can be used to protect different web applications. The main advantage is that it is easy to deploy and capable of adapting to changes in the web application.

However, the detection capabilities provided by the one-class approach do not adapt so well to prevent both zero-day attacks and attacks that exploit specific vulnerabilities of a application, in particular those involving suspicious input. Thus, as a complementary solution, we have experimented with techniques that use high-order n-grams up to some n to provide anomaly detection capabilities based on the expected language signature of the input fields of the application. The n-gram approach requires to have an application specific dataset with valid request to train its model. By modeling the language signature of each attribute of the web application it is possible to recognize known attacks with good precision and also to detect zero-days attacks.

In summary, if we need to fast deploy a WAF to protect a web application without having a specific dataset or the web application changes constantly (e.g. public website powered by a content manager), we propose to use the one-class approach. We will call this Scenario I, when we want to protect a web application without having any specific training data. In this case we would like to address the following question: *Is it possible to build an attack detection system learning from training data collected from other web applications?* A second question in this scenario it is related to MODSECURITY: *Can we improve the results of* MODSECURITY *using machine learning methods?* That is, can we reduce the number of false positives (FP) generated by MODSECURITY? In the scenario where we need to protect a business critical web application where high levels of security are required, and the application's changes are controlled, we can deploy the n-gram approach. In this second scenario, Scenario II, it shall be required to have a testing phase before each new deployment of the application, so specific training data will be available. This training data shall be used to train and update the n-gram model before the application's go-live. In this scenario, *we would like to understand the attainable performance of machine learning methods against that of* MODSECURITY.

III. LEARNING MODELS

We now turn to the description of the models proposed for enhancing the capabilities of MODSECURITY using machine learning techniques. These models follow a common pattern that we proceed to describe.

1) Training set.: In web applications information between the client and the server is exchanged using the HTTP protocol. This protocol is based in a request/response model using plain-text (ASCII) messages. The model is built up from a training set $\mathscr{T}$ of normal HTTP requests that have been previously recorded. The set $\mathscr{T}$ is assumed to be representative of legitimate traffic. An HTTP request is just a string with the following structure: a request header (specifying the method to be applied, the URI on which it is applied and the protocol version), a collection of header fields of the form *field=value* and possibly a request body.

2) Pre-processing: When non-ASCII information has to be exchanged between the client and the server the same encoding has to be employed. Different types of encoding might be used (e.g. URL encode [6]). Before building the statistical model, the requests are first pre-processed to decode the information they contain. This might also involve parsing part of the HTTP request structure or removing parts of it that are considered useless for the model. How granular is the parsing processing of the request structure constitutes a first choice in the model design.

3) Feature extraction: Then, a collection of features are extracted from the request. These features are related to the occurrences of a collection $\mathscr{A}$ of substrings or TOKENS in the request. The criteria used to select these tokens is a second design choice of the model. We experimented with two different criteria: i) using a fixed collection of words determined by a security expert from the current state of the art in web attacks, and ii) automatically computing them from the training set (n-grams) . This gives rise to an internal representation of the requests in terms of the numbers resulting from this counting. Such representation can be conceived as a vector of numbers, where each position of the vector contains the number associated to a given token, or more generally, as a mapping associating numbers to tokens (bag of words).

4) Model computation: The model is a probabilistic distribution for the elements of the internal representation estimated

from those computed for the requests in the training set $\mathscr{T}$. The algorithm to be used for computing this distribution is a third choice to be made in the model design. This distribution provides the *signature* of the web application, which characterizes the normal HTTP requests exchanged with the application. From a geometrical perspective, the model can also be conceived as a hyper-volume containing all the points (vectors) corresponding to the internal representation of a normal HTTP request.

5) Request classification: The WAF uses a model M to analyze incoming HTTP requests on-line, one by one, as they are received, and classify them into two classes: normal and abnormal requests. Abnormal requests are supposed to be an attack, and hence processed differently according to the WAF policy (logged, filtered, etc). In order to test a given HTTP request r, the WAF computes the internal representation of r. Then, a *score* $s_r = dist(r, M)$ is computed for the request, using a distance function *dist* which measures how far the actual signature of r is from the expected signature provided by the model M. A score of 0 means that the HTTP request contents completely matches the expected signature. The higher the score, the less the field contents meets the expected frequency distribution. Another design choices are the distance and the criterion $\mathscr{C}(M, r)$ to be used for deciding whether the distance to the signature characterizing the application is far enough to deem the request r as anomalous. Usually, this criterion relies on the distribution of distances that can be drawn from the training set $\mathscr{T}$, by computing the score of each individual training request.

The rest of this section is devoted to describe how the general pattern just described has been instantiated into two different models: one-class classification and n-gram analysis.

A. One-Class Classification

One-class classification [7] assumes that training information is only available for one of the classes; in our case the class of valid requests, this is also called *novelty detection*[8]. Hence, the problem of one-class classification is to learn the behavior of valid requests and identify attacks as deviations from the learned behavior for the valid class. That is, the goal is to define a boundary around the valid class maximizing the number of valid requests inside the boundary and minimizing the number of non valid instances inside it. Valid instances that fall outside of the defined boundary are false positives of the classifier.

To build such a classifier we must define the features that are supposed to capture the properties of common known web application attacks (e.g. SQLINJECTION, XSS, among others). We shall rely on the the experience of a security expert to do that. In III-A2 we describe the features that will be used to characterize the request.

1) Pre-processing: In this phase, besides performing decodification, we also filter the request headers that are used to exchange contextual information between the user-agent and the server. All information contained in headers that are specific to the protocol, such as cookies, proxies and IP, which do not represent user behavior and should not be considered to infer application behavior, are filtered out.

Table I
SELECTED SPECIAL WORDS BY THE SECURITY SPECIALIST

<	../	alert	exec	password
<>	’	alter	from	path/child
<!–	“	and	href	script
=	(	bash_history	#include	select
>	)	between	insert	shell
|	$	/c	into	table
||	*	cmd	javascript:	union
-	*/	cn=	mail=	upper
–>	&	commit	objectclass	url=
;	+	count	onmouseover	User-Agent:
:	%00	-craw	or	where
/	%0a	document.cookie	order	winnt
/*	Accept:	etc/passwd	passwd	

2) Feature Computation: Several known attacks make use of specially crafted inputs to force the server behave in a non expected manner. Most of these inputs use as part of the attack special characters, like for instance . , ; < >, or special substrings like for instance *select*, *alert*, *passwd*. We propose to use these special characters and substrings to capture the characteristics of the request. This is basically a bag-of-words model where each document (in our case each request) is represented as a bag of its *words* (the special characters and substrings mentioned above). The bag-of-words feature vectors are calculated counting the appearance of each word in the request. In this way, each request r is represented with a vector x_r where each element in this vector counts the number of times each word is observed in r. To select the set of words, we studied valid and attack requests, and applied the knowledge of a security expert. In Table I we list all the words used. Since we are modeling the valid class, we expect that few of those words will be present in each request. As we are focused on user input, we analyze the query string of the URL, the request body and the requests headers, all information that could be modified by the user of the application.

3) Model computation: The idea is to detect attacks as deviations from normality using a one-class classifier; a request will be classified as an attack if it is far from the observed valid class. A well known method to build a one-class classifier is to estimate the *p*robability density function (pdf) of the training data and to estimate a threshold to segment normal and abnormal instances. Among the existing methods to estimate the probability density function, Gaussian Mixture Models (GMM) is one of the most common one. If x is a feature vector, it is assumed to be an observation of a random variable X with a pdf:

$$p(x) = \sum_{k=1}^{n} P_i \mathcal{N}(x; \mu_k, \Sigma_k), \tag{1}$$

where n is the number of gaussians, P_k, μ_k and Σ_k the weight, mean vector and covariance matrix of component k respectively and $\mathcal{N}$ a Gaussian pdf. The Expectation Maximization (EM) algorithm [9] can be used to estimate the parameters of the GMM (mean vector μ_k, covariance matrices Σ_k and weights P_k). In our case, each component of the obtained GMM constitutes a cluster that captures the distribution of valid requests. We also use EM to estimate n, the number of components (clusters).

Once we have the GMM of valid class, in order to classify an instance into valid or attack, we need to compute the distance to each component of the GMM (cluster) and apply a threshold on this distance.

In this work we use the Mahalanobis distance since it measures the distance of a feature vector to a distribution. In this case, the distribution corresponds to a component of the GMM (C_k), so the distance is defined as:

$$dist(x_i, C_k) = \sqrt{(x_i - \mu_k)\Sigma_k^{-1}(x_i - \mu_k)} \quad (2)$$

If x_i correspond to the mean of the distribution then the distance is 0, and increases when the point moves away taking into account the standard deviation of the distribution. If one of the dimensions has a standard deviation of 0 in the distribution, the Mahalanobis distance could not be calculated. For this reason, we adjust the covariance matrix by adding a regularization term to it $\epsilon * Id$, where ϵ is the smallest standard deviation in $diag(\Sigma_i)$ different from 0 and Id is the identity matrix. Intuitively this regularization allows the observation of new values on requests not seen in $\mathscr{T}$.

4) Request Classification: In this approach, the score s_r is a vector where each position corresponds to the $dist(r, C_k)$. Before being able to do request classification, we need to estimate the threshold of each cluster. The threshold of the cluster is defined by analyzing the distribution of the distances of each instance of $\mathscr{T}$ that where assigned to the cluster. This transforms the multi-dimensional space of features into a real number corresponding to the distance of the request to the the cluster. Given that all requests assigned to a cluster (as a result of using EM) can be represented by a component of the GMM, we approximate the distribution of teh distances with a normal distribution. We define $\overline{dist}_k$ to denote the mean of all distance of x_i to C_k where $x_i \in C_k$ and std_k to denote the standard deviation of the distances of $x_i \in C_k$. Finally, the threshold t_{C_k} is defined as follows:

$$t_{C_k} = \lambda[\overline{dist}_k + 10 * std_k], \lambda \in (0, 1] \quad (3)$$

The use of $\overline{dist}$ and *std* makes it possible to have in the model a specific threshold for each cluster that depends directly on how the instances that belong to that cluster behave. The constant factor 10 was derived using different dataset in a way that when $\lambda = 1$ the distance for all $x_i \in C_k$ is less than t_{C_k}. Varying the threshold, by multiplying it by a constant λ, let us change the model's precision. Even if there exists a specific threshold for each cluster, the λ constant is the same for all clusters. This allows us to have only one parameter to control the precision of the whole model. As λ grows more valid requests fall into the clusters, but also more attacks. When λ decreases, the false positive rates increase, but more attacks are missed. This parameter allows us to have different operational points.

Before deploying this approach to protect a web application, we need to define which operational point to use. The best scenario corresponds to have a labeled dataset with valid requests and attacks. In this scenario is possible to adjust this threshold to its best operational point. In real life applications, our best scenario is having examples of only normal behavior. In this case, one metric to define the operational point could be the amount of false positives that we are willing to accept. The worst scenario is when we do not have even an application specific dataset. For these scenario, we propose to use as training dataset a mixture of several datasets from different web applications. In Section IV we will present the results of this approach (Scenario I) to show that good performance scores can be achieved.

After training the model, the classification of a new request r starts by calculating the score s_r. This results in a vector where each position corresponds to the Mahalanobis distance of r to C_k. The criterion $\mathscr{C}(M, r)$ for this approach corresponds to compare the score s_r with the corresponding threshold for each cluster. If any of the scores is lower than the threshold the request is classified as normal, otherwise is classified as an attack.

B. Anomaly detection using n-grams

We now turn to present our second approach, which positively characterizes the normal behavior of each application using n-grams as tokens.

A well-known technique for identifying the language of a piece of text is to measure the frequency of the n-grams occurring in the text. An n-gram is a sequence of n (usually consecutive) symbols of the alphabet used in the text (for example, ASCII characters, words, etc.). For instance, the most frequent character trigrams in Spanish are *del* and *que*. On the other hand, the character trigrams *whe* and *ike* are very rare in Spanish, but not in an SQL sentence used for an SQL-injection attack, as they appear in the keywords WHERE and LIKE. In the most general case, the n-gram may be a sequence of words instead of characters (in which case we may consider the symbols to be the words, and the rest of the analysis remains the same).

1) Pre-processing: The collection of all the n-grams ocurring in a text can be computed following a very simple, fast and linear algorithm, in which a window of size *n* is slided all along the text. This algorithm may be preceeded by a *tokenization procedure*, in which the text is chunked into *words*, separed by a set of *delimiters*. A delimiter is any sequence of symbols matching a given regular expression. In the case of an URI, for example, the slash symbol (/) and the ampersand symbol (&) are natural delimiters of an URI, and the n-grams are sequences of strings, each one describing either the domain of the web site or a resource in its ressource hierachy. For those languages where there are no clear delimiters in the text, words are just the alphabet symbols themselves. Identifying word with symbols provides a tokenization procedure that can be applied in any case, as it is independent from the language grammar. This is an important advantage, as the same method can be uniformelly applied to all fields. Moreover, in practice web applications frequently make use of custom HTTP fields, and their structure is unknown.

Not all n-grams are equally relevant. Actually, some distinctions may be more problematic than helpful. For example, IP address 168.192.0.1 does not better characterize valid requests

than IP 168.192.0.2. On the other hand, making the model to be dependent on the occurrence of particular IP addresses is more prone to overfitting with respect to the training set $\mathscr{T}$. For this reason, we count the n-grams obtained after applying some *abstraction function* $abs : S^m \to S^k$ on them for some $k \leq m$, where S is the ASCII alphabet. In our experimental results we use an abstraction function performing the following transformations on the decoded HTTP request: (a) letters are uncapitalized, (b) accents are removed and (c) digits are replaced by the capital letter "*N*".

Opposite to the previous approach, in this case the preprocessing step tries to exploit as much as it can from the HTTP structure. Even if the HTTP protocol does not impose any structure on the request body, most of the interaction with the user consists in presenting HTML forms to be filled. Once filled, HTML forms are transmitted to the server as a list of parameter assignments $\mathrm{y}_1 = \mathrm{v}_1$ & $\ldots \mathrm{y}_k = \mathrm{v}_k$ usually separated by the & symbol. Therefore, the request body may be either plain text or a parameter assignment. In this latter case, we do not work directly on the request string itself as in the one-class approach: we first parse its structure, keeping the HTTP field and parameter contents, and discarding their names. We use the term *model fields* to design either a header field, or a web application parameter. The model computes a separate signature for each model field.

In the sequel, we introduce a general framework for experimenting with attack detection based on n-gram frequencies. We consider n-grams of length m formed from symbols from the alphabet S, for all $m \leq n$, where n is a model parameter that can be adjusted for each application.

2) Feature extraction: Formally, a model field is a pair formed by an HTTP request field name and either a parameter name y_i (parameter assignment), or a special value $\perp$, otherwise (for representing unestructured, plain text body). The basic *attributes* of the model are the n-grams that can be found in the model fields contents. Each attribute is a pair $(x, z) \in \mathscr{A}$ formed with a model field x and an n-gram $z \in S^m$ ocurring in x. In the sequel, we use the notation $x.z$ for this pair. The model is enriched with an additional attribute not related to n-grams, namely, the number of characters (length) of each model field x. This additional attribute is also a good indicator for code injection attacks, as they are likely to increase the expected field length.

We associate a random variable X_a to each attribute $a = (x, z) \in A$, which measures events related to the occurrences of the n-gram z in the model field x. We focus on two types of events: the number of occurrences of the n-gram in the model field (integer value) or its frequency inside the field contents, that is, the number of occurrences of the n-gram divided into the total amount of n-grams occurring in this model field (real value). Measuring the number of n-grams is better suited for those model fields having an enumerated type of possible values, for example, the languages that the web application support. Measuring the n-gram frequency performs better when the field length may significantly vary from one request to another. For example, a message field in a contact form of the web application contains free text that may be arbitrarily long. The number of occurrences of a given symbol may be quite different depending on the message, but the frequency of that symbol should be rather the same in all messages.

A *distribution* for one of these random variables X_a is a characterized by a tuple $d = (\mu, \sigma, max, min, N_d, H_d) \in \mathscr{D}$, where μ is the distribution mean, σ^2 the distribution variance, $max, min \in \mathbb{N}$ the maximum and minimum values that were sampled for X_a, and $N_d \in \mathbb{N}$ the number of sampled values. $H_d : \mathbb{R} \to \mathbb{N}$ is the histogram counting the number of requests in $\mathscr{T}$ having a given frequency for a.

3) Model computation: We assume that each random variable X_a is independent from the other random variables. This means that constructing the probabilistic distribution of the internal representation of the HTTP requests amounts to construct one independent distribution for each attribute $a \in \mathscr{A}$. The distribution parameters of each random variable X_a are incrementally approximated using Welford's algorithm for online computation of its mean and variance [10]. Therefore, the result of this iterative process is a map $M : \mathscr{A} \to \mathscr{D}$ representing the learned distribution of the random variable X_a associated to each attribute a. This mapping provides the *language signature* of each model field x, in terms of the probability distribution of the n-grams in the specific language of x. We compute a specific language for each field. This is an important difference with respect to other works [11, 12], which analyze the traffic at lower (TCP) layers: while each field represents a logical unit of the web application, with its own specific laguage (numbers, addresses, names, credit card numbers, etc.), this structure is splitted and mixed at the TCP layer into one single language (the union of all these specific languages).

4) Request classification: In order to test a given HTTP request r, the WAF computes the map $f_r : \mathscr{A} \to \mathbb{R}$ of concrete values for each attribute of r. Then, each model field x of r is considered. If field x is not defined in model M, then r is rejected. This amounts to say that any request containing unknown HTTP fields or web application parameters that have not been met during the training phase is deemed as anomalous. Otherwise, the *score* $s_{r.x} = \sum_{i=1..n} dist_i^x(r, M)$ is computed for field x, using a function $dist_i^x$. The definition of the distance depends on the n-gram length. Also, different distances could be associated to different fields. If there is a field x for which the obtained score does not satisfy some given criterion $\mathscr{C}$, the whole request is considered valid, otherwise it is deemed as anomalous (outlier value). The criterion used for determining whether the distance is acceptable is that the score falls inside the rank defined by the minimum and maximum values of the score distribution *ds* associated to the requests in the training set $\mathscr{T}$, that is, $\mathscr{C}(ds, s_r) = ds.min \leq s_r \leq ds.max$. This distribution is drawn from the training set $\mathscr{T}$, by computing the score of each individual training request.

The distance from the request signature to the language signature is measured using Mahalanobis distance under the assumption of independence of the random variables associated to the tokens:

$$dist_i^x(r, M) = \sqrt{\sum_{z \in \mathscr{T}_i^x} \frac{|\mu_M(x.z) - f_r(x.z)|^2}{\frac{1}{|\mathscr{T}_i^x|} + \sigma_M^2(x.z)}} \qquad (4)$$

Mahalanobis distance measures how many standard deviations away the request signature is from the language signature. As a consequence, differences for n-grams with highly variable frequencies do not weight as much as differences for n-grams with a rather constant frequency. The constant $\frac{1}{|\mathscr{T}_i^x|}$ is added to the denominator to prevent a division by zero for those n-grams with constant frequency. Note that the larger the number of samples for x's i-grams is, the greater the difference from the mean weights in the total score.

In practice, the training set $\mathscr{T}$ is a small fragment of all possible HTTP requests that the web application accepts, so a given n-gram of the test request r may not be present in $\mathscr{T}$, but still be a valid input for the web application. This is problematic, as unknown n-grams are assigned the highest possible anomalyness score. However, most of the fields contain values that are a subset of some larger language. For instance, a form field in the web application corresponding to an address in Madrid is a particular case of a piece of text written in Spanish. Such larger language has its own language signature P_x, which can be used as a *prior distribution* for the field x. By assigning priors to some fields we decrease the false positive rates: should a given n-gram not be defined in M for that field, we use that prior as the expected distribution for the attribute. Notice that this approach is possible when the analysis is performed at the HTTP level, when the application fields can be recognized, but is much harder to implement at the lower communication levels. If the n-gram is neither defined in this prior, then we assign the singleton distribution d such that $\mu = \sigma = 0 = min = max = 0$, $N_d = \mid \mathscr{T}_i^{x.z} \mid$ and $H_d = 0 \rightarrow \mid \mathscr{T}_i^{x.z} \mid$, which represents that 0 is the expected frequency for this n-gram according to the training sample $\mathscr{T}$.

In some situations, it may be helpfull to deliberately exclude some fields from the model. For example some web applications generate a dynamic URI or encode dynamic information as part of the URI itself. Those URI have almost no chance of being met during the model training, and therefore are prone to generate false positives. In such cases, excluding the URI from the analysis improves its performance. Another example that can be excluded are those fields which are supposed to be close to random data, such as a parameter transmitting a passowrd or encrypted data. In our implementation, a configuration file enable to assign the model parameters to be used in the model construction, such as the n-gram length bound n,the tokenization method and its delimiter, the distance, the model fields to be excluded from the analisis, etc. Independent values can be assigned for these parameters for each model field.

IV. Experimental results

We have evaluated the proposed models experimenting with three datasets. We have privileged the use of public datasets which are intended to be used for analyzing web application attacks. Unfortunately, there exist very few datasets of HTTP traffic in general, and even fewer with tagged attacks. The only public datasets that seemed useful for our purpose were the CSIC2010 dataset [13] and the dataset from the ECML/PKDD2007 challenge [14]. We discarded the use of the DARPA 1999 dataset, as it contains TCP traffic and is no longer representative of the state of the art in nowadays attacks [15]. We also evaluated the models on a dataset of our own which was obtained from the HTTP traffic to the web server of a University. We will refer to this dataset as DRUPAL.

Each of the used datasets is made of a collection of complete HTTP requests (header and body) which have been partitioned into one training dataset and two testing ones: one for valid traffic and another for anomalous one. In Table II, we present the details of each dataset. As explained before, our models only use normal traffic for training, labeled attacks will be considered only for evaluation purposes. The model performance was measured with the usual true positive rate (TPR) and true negative rate (TNR) indicators. The TPR corresponds to the percentage of anomalous requests that are detected as attacks. The TNR corresponds to the percentage of valid request that are spotted as normal traffic. The results are presented as ROC curves that describe how these two indicators are affected by the parameters of the models.

The baseline to which we compare the behavior of our models are the TPR and TNR values obtained as the outcome of using MODSECURITY configured with the OWASP CRS version 2.2.9 out of the box.

In the one-class classification approach the main parameter of the classifier is the threshold that governs the size of the clusters. The size is adjusted by a parameter λ whose values range from 0 to 1 (see equation 3). Each value of λ determines an operational point of the classifier. For each dataset we shall present: i) the ROC curve obtained for the one-class classifier when training is performed using specific data for the web application and varying λ in $(0, 1]$ (blue curve). This ROC curve can be seen as the ideal result for the classifier since uses an application specific training dataset, ii) the ROC produced with the one-class classifier trained with data from other web applications will be plotted as a green thicker curve. This result addresses the main question in Scenario I. The black diamond over the green curve indicates a default operational point obtained with $\lambda = 0.5$. This point was fixed based on the number of FP for the training set. Since in the Scenario I we do not have specific training data we cannot fine tune the operational point, iii) the result of MODSECURITY using the OWASP CRS out of the box is indicated with a blue square, iv) the yellow ball, which shows the performance of the model that combines one-class with MODSECURITY following the combination strategy discussed in Section II-C

In the n-gram model approach a fine tuning of the model was experimented, assigning different tokenization methods, distances, prior distributions and n-gram length bounds to each model field. For the sake of simplicity, we restrict the presentation of the results for the n-gram length bounds $n = 1 \ldots 5$ when an optimized configuration is assigned to the other model parameters.

A. CSIC2010 *dataset*

In year 2010 the Spanish Research National Council (CSIC) developed and made available a dataset to test a system to protect web applications [13]. This dataset embodies a collection of normal and abnormal HTTP requests for a web application

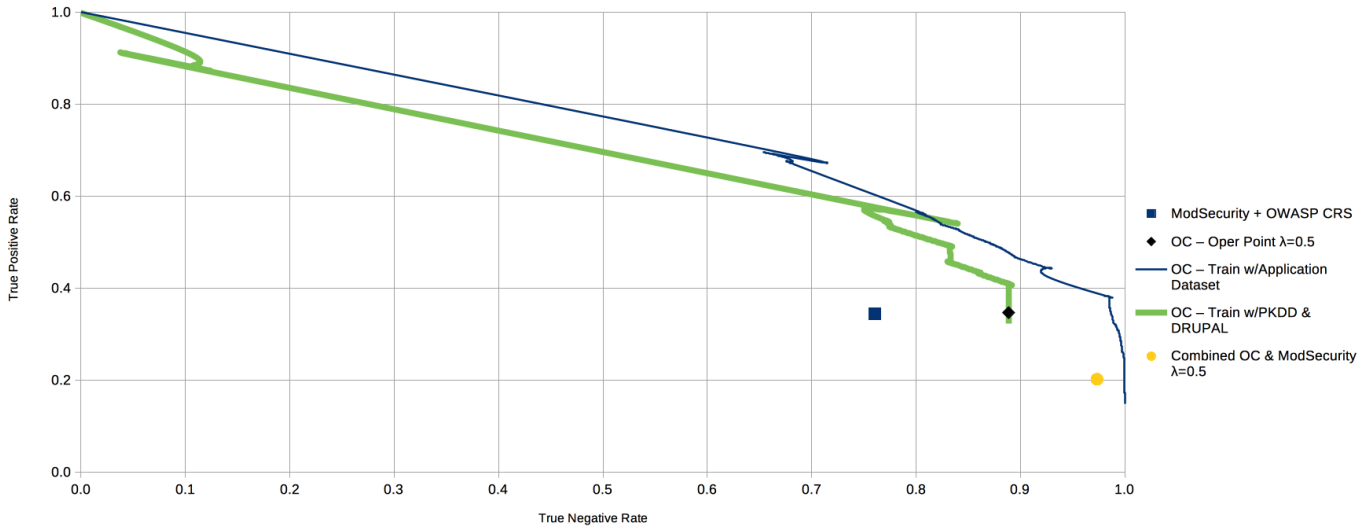

Figure 1. Results for the CSIC2010 dataset. Blue square: MODSECURITY with OWASP CRS out of the box. Blue solid curve: one-class varying λ (OC). Green thicker solid curve: one-class varying λ - training with mixed dataset. Black diamond: operational point for OC. Yellow circle: OC operational point combined with MODSECURITY.

Table II
DATASETS COMPOSITION

	Train (Valid)	Test (Valid)	Test (Attack)
ECML/PKDD	24504	10502	15110
CSIC	36000	36000	25065
Own Dataset	45907	19675	1287

that provides functionalities to perform on-line shopping on a tiny store. The dataset is made of 36.000 valid requests for the training set, another 36.000 different valid requests for testing normal behavior, and 25.000 abnormal test requests mixing attacks with valid requests containing infrequent characters in the parameter fields (typos). Unfortunately, the distribution between attacks and infrequent values is not specified. The attacks were generated using tools such as Paros (which later became OWASP ZED [16]) and w3af [17], and include SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, server side include, parameter tampering, among others. Another important characteristic is that it was conceived and intended for this kind of experiment: the HTTP requests have a few fields which always take a limited collection of values. It contains several duplicated cases, as each GET request in the dataset containing a query in the URI is followed by an equivalent POST request, with the same query in the body field. The attacks are concentrated on the web application parameters.

In Figure 1 we present the results in terms of TPR and TNR for the CSIC2010 dataset. The simplicity of the normal requests and the mixture of attack with just anomalous traffic is observed in the high TNR and the low TPR obtained by MODSECURITY (blue square). As can be noticed, the one-class classifier trained with data from other web applications (green curve), used to evaluate Scenario I, produced good performance scores (TNR and TPR). The ROC curve shows that there are several points that outperform MODSECURITY (blue square). Furthermore, if we compare the results obtained using an application specific training set (blue curve), we can see the performance it is not far from the ideal one (when we train the one-class classifier with data from the same web application). Finally, the yellow ball shows the performance of the model that combines one-class with MODSECURITY. The combination improves in terms of TNR but decreases in TPR. This is because our main objective is to decrease MODSECURITY false positives so the combination algorithm only mark a request to be an attack if both experts tag it as an attack. This means that some requests that were tagged as an attack by the one-class model where tagged as valid by MODSECURITY and viceversa.

If we compare the results with our baseline, e.g. for the same TNR, MODSECURITY detects around 34% of attacks, where the one-class approach detects around 56%. In this particular dataset, the integration of MODSECURITY rules with the one-class approach does not improve the results of one-class by itself in terms of TPR but clearly reduces the number of false positives (i.e., TNR close to 1). See Table III for details on the results.

For the n-gram analysis, we configured our tool to perform some specific behaviors for some of the model fields. Despite the default n-gram length, the analysis is always bounded to monogram analysis for URL, login identifier, customer's national identifier and passwords. This is because the only biased aspect of those fields is the set of allowed characters, but almost any combination of them is possible in principle, so higher order n-gram analysis is prone to produce false positives. In the case of the URI, the slash bar character (/) is specified as delimiter, and monogram analysis is performed on the number of resulting words that occur in the field, not its frequency. The reason is that is quite unlikely that one of these words (which represent resource folders in the web application directory) appears more than twice in the URI. Finally, following the technique explained in section

Table III
TRUE NEGATIVE AND TRUE POSITIVE RATES (IN %) FOR EACH DATASET

Method		CSIC2010		DRUPAL		ECML/PKDD2007	
		TNR	TPR	TNR	TPR	TNR	TPR
ModSecurity		76,1	34,3	61,1	72,2	42,8	93,0
One-class: $\lambda = 0,5$		88,9	34,6	93,3	86,2	0,0	98,6
Combined OC-MS		97,3	20,1	99,1	63,0	42,8	92,1
N-grams	n=1	99,9	93,0	93,9	95,9		
	n=2	99,9	94,8	94,4	97,6		
	n=3	99,5	96,1	92,0	97,5		
	n=4	96,2	96,8	90,7	98,8		
	n=5	90,9	97,5	89,4	98,9		

III-B, we specified a prior n-gram distribution for the fields corresponding to names, cities and addresses, drawn from a collection of Wikipedia articles written in Spanish. The use of priors for these fields reduced the false positive rate in 3% for trigrams. The most significant impact on false positives is observed for bigrams and trigrams.

B. ECML/PKDD2007 *dataset*

In year 2007 the 18th European Conference on Machine Learning (ECML) and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), put forward a challenge on Analyzing Web Traffic [14]. The challenge objective was to construct an algorithm based on machine learning techniques to perform multi-class, contextual classification and attack pattern isolation. As part of the challenge it was provided a dataset which contained requests classified in seven different types of attacks plus valid requests. The dataset is composed of 35.006 requests classified as normal and 15.110 requests classified as attacks.

After analyzing the dataset we infer that some process of obfuscation/anonymization was executed on the dataset to protect urls, parameter names and values. Those items were replaced by random values and the obfuscations process does not seem to be an injective function, as no URI, parameter name or value appears twice in the dataset. As a consequence, the ECML/PKDD2007 dataset revealed useless for carrying on n-gram analysis experiments, as the valid behavior that could be used for training purposes consists in random data with no bias. On the other side, the one-class approach can be successfully used to detect attack signatures. Figure 2 shows the result of this experiment. Regarding the Scenario I (green curve), the performance evaluated with TNR and TPR is not as good as in the case of CSIC2010 and DRUPAL dataset that will be presented below. After careful evaluation, we observed that when the process of obfuscation is executed in the dataset the random values were generated using letters, numbers and the $=$ and $-$ symbols. These two last symbols are included in our list of tokens defined by the expert. After analyzing the problem, we found that this artificial way of generating the values are so specific and different from the normal traffic to the web application, that the generic approach did not work as expected. On the other hand, if we compare the results obtained using a model trained with specific requests (blue curve), we observe that good performance is achieved. In the curve there are some points with less FP than ModSecurity but slightly worse performance when considering TPR. We believe that the way this dataset was generated strongly conditions the results. Better results should be obtained if we removed the features mentioned above.

C. DRUPAL *dataset*

The CSIC2010 and ECML/PKDD2007 datasets have been artificially conceived for the sake of experimenting with machine learning techniques. In order to evaluate our approach on real life applications, based on actual requests and attacks, we crafted a dataset by registering three days of incoming traffic to the public website of a University [1]. The only post-processing of this dataset consisted in blurring password values in the request.

Since the requests are from real traffic, this dataset is less balanced: it has 65582 valid request and 1287 real attacks. It also contains an important amount of custom HTTP fields used by the applications hosted in the web site.

The approach of registering traffic in order to exploit it as a dataset has been already used in several previous works. One of the difficulties that raises is classifying the received requests into valid ones and attacks, so that the dataset can be separated into training and a testing part. The web site of the University is protected by an instance of MODSECURITY featuring the OWASP CRS, which has been tuned for several years by a team of security and infrastructure experts. We therefore used MODSECURITY as the labeling tool: those registered requests that were accepted by MODSECURITY were considered as the valid traffic and used for the training step and for the testing dataset, while those requests that MODSECURITY rejected were used as the testing dataset for the attacks.

Figure 3 describes the results for the DRUPAL dataset. We observe that they are similar to the ones obtained for CSIC2010. The one-class classifier trained with data from other web applications (green curve), used to evaluate Scenario I, is very close to the blue curve which is the ROC generated with the one-class classifier trained with application specific data. The default operational point, black diamond, outperforms MODSECURITY, and several of the points in the previous curves clearly perform better than MODSECURITY. Finally, the results of the model that combines one-class and MODSECURITY (yellow ball) improves in terms of TNR but decreases in TPR. This is because our main objective is to decrease MODSECURITY FP so the combination algorithm only marks a request to be an attack if both experts tag it as an attack. This mean that some request that where tagged as an attack by the one-class where tagged as valid by MODSECURITY and viceversa.

D. Discussion

Based on the results for the three datasets and the two proposed approaches we discuss in what follows the main questions presented in Section II-C.

[1]The dataset is not public, but it is available on demand to other researchers willing to reproduce our results or comparing them against other ones, by writing to the authors

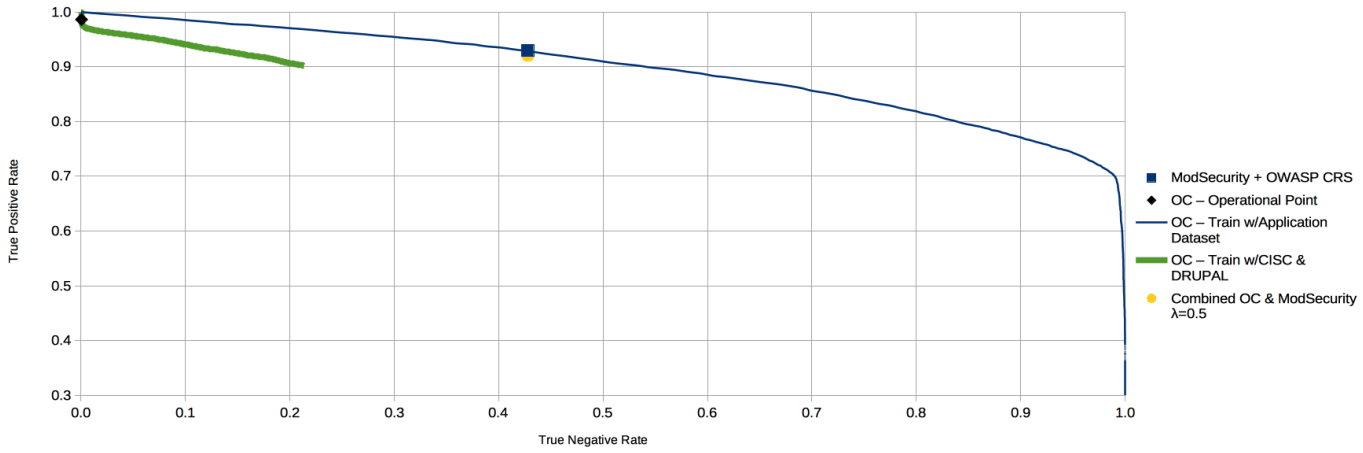

Figure 2. Results for the ECML/PKDD2007 dataset. Blue square: MODSECURITY with OWASP CRS out of the box. Blue solid curve: one-class varying λ (OC). Green thicker solid curve: one-class varying λ - training with mixed dataset. Black diamond: operational point for OC. Yellow circle: OC operational point combined with MODSECURITY.

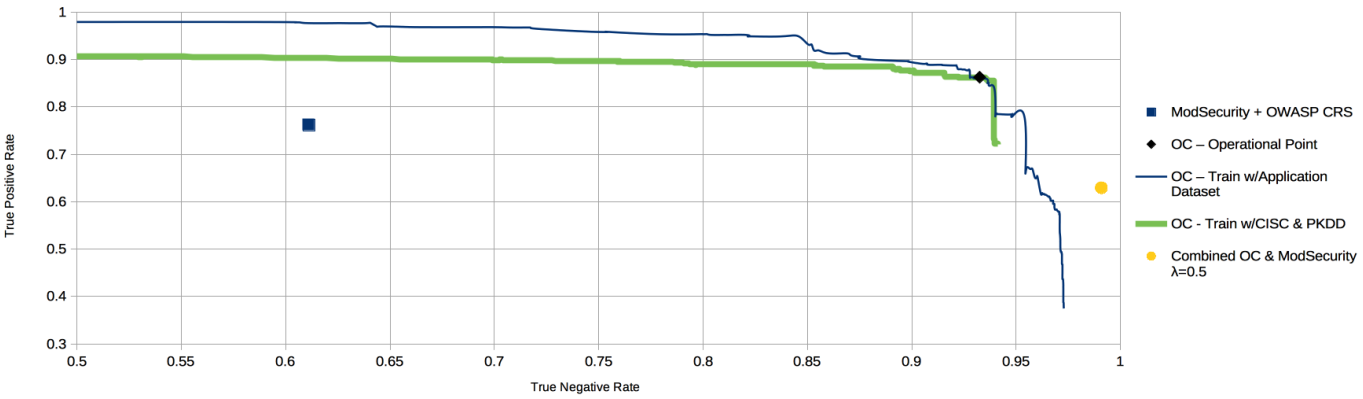

Figure 3. Results for the DRUPAL dataset. Blue square: MODSECURITY with OWASP CRS out of the box. Blue solid curve: one-class varying λ (OC). Green thicker solid curve: one-class varying λ - training with mixed dataset. Black diamond: operational point for OC. Yellow circle: OC operational point combined with MODSECURITY.

a) Scenario I: Is it possible to build an attack detection system learning from training data collected from other web applications?: The results for datasets CSIC2010 and DRUPAL present evidence that it is possible to build a one-class classifier using generic training data, that is, using a dataset with request not from the web application to protect. Furthermore, the degradation with respect to the same classifier trained with specific data for the web application is not critical. For the case of the ECML/PKDD2007 dataset we already discussed the particularities which justify the differences between the one-class classifier trained with specific and non specific training data.

b) Scenario I: Can we improve the results of ModSecurity using machine learning methods?: Looking at the results for the datasets CSIC2010 and DRUPAL we can conclude that it is possible to improve the results of MODSECURITY. For the ECML/PKDD2007 dataset the results of the method that combines MODSECURITY and one-class are the same as MODSECURITY. One advantage of one-class is that we have several operational points to choose from. In particular, we can reduce the number of FP (increasing TNR) but reducing TPR. As mentioned above, the particularities of the dataset prevent taking advantage of the one-class classifier trained with generic data. For future work we plan to analyze this issue in detail.

c) Scenario II: Attainable performance of machine learning methods against ModSecurity: Based on the obtained results we can draw the following conclusions. First, based on the results of one-class and n-grams, it is possible to improve the results of MODSECURITY using machine learning. Second, looking at the TNR and TPR scores computed for the n-gram method, we can say that the n-gram approach is a good solution if we have an application specific dataset to train its model.

V. RELATED WORK

There exist in the literature several previous works concerned with the application of machine learning techniques to detect information systems attacks.

A. Virtual patching

Web applications are designed using three major tiers: a thin client (web browser), the business logic and a datastore. Virtual patching may be applied to any of those tiers. In recent

years work has been reported [18, 19, 20] concerning the use of database firewalls (DBF) as remediation tools. In all of these works anomaly detection techniques have been applied to analyze the flow of SQL sentences from the business logic tier of the application to the database with the objective of detecting (only) SQL injection attacks. As a DBF can only inspect the traffic between the business logic and the database server it is not capable of detecting attacks that are not directed to the data layer of the application.

There exists several proprietary implementations of WAF technology. They have been reviewed and ranked in a recent report of the Garner Magic Quadrant [21], where, in particular, one can find the following statement concerning the reviewed technology: *"Use of machine learning is rare and often still unproven"*.

B. Application layer analysis

One of the seminal works in anomaly detection techniques was developed by Wang and Stolfo [11], where they introduced PAYL, a payload-based anomaly detector for intrusion detection. Their approach consists in comparing the byte frequency distribution in network packets using (a simplified) Mahalanobis distance. Accordingly with the current attack vectors of that time, their objective was focused on protecting an internal network from packets carrying worms or other forms of malware. Consistently, their model works on TCP packets, that is, on the network OSI layer. As the byte distribution heavily depends on the application protocol, they divide the stream into smaller groups of packets, grouping them by destination port and by payload length. They applied PAYL to the traffic to port 80 (HTTP) of the 1991 DARPA IDS dataset, obtaining excellent detection results. However, their results relate to the attack vectors of 1999. Indeed, Ingham and Inoue report in [15] that the DARPA IDS 99 dataset only contain four web attacks. Moreover, it does not contain any of the OWASP TOP 10 attacks [22][2]. As a consequence, while [11] reports on a 100% detection rate for traffic on port 80 of the DARPA IDS dataset based on byte frequency distribution of the unparsed TCP payloads, performing the same analysis on the CISC2010 dataset yielded a detection rate of only 40% of the true anomalous cases. This low performance has been also independently reported by Ingham and Inoue in [15]. The reason is that token distribution varies from one HTTP field or web application parameter to another, which are spread along several TCP packets. Protecting web applications requires shifting the analysis to the application OSI layer, focusing on HTTP requests.

C. Machine learning techniques

a) Training from normal traffic only: In the ECML/PKDD2007 challenge [14], the objective was to classify web application requests using a multi-class approach, in which the training dataset contains labeled HTTP requests which were classified into normal ones and attacks. Several solutions were presented [24, 25, 26]. In particular, Gallagher et al [26] presented a solution that used classical techniques of information retrieval. Even if their solution performs quite well it is restricted to detect only known attacks and relies on a labeled dataset which contains dozens of thousands of requests. In our case, our solutions only require a training dataset that contains normal traffic. In addition to this, we have shown that the one-class approach works quite well even in the absence of a dataset proper of the application being protected. The scenario in which the WAF is trained with a dataset containing requests from applications other than the one being protected our results outperform MODSECURITY with the OWASP CRS out of the box. In [27], Raïssi et al conclude, based on the results of the ECML/PKDD2007 challenge and a feedback survey written by the challengers, that using machine learning techniques to detect web application attacks requires to involve the security experts earlier in the knowledge discovery process. In the one-class approach we have pursued, the feature selection phase uses the knowledge of the security expert to identify the tokens to be considered in the model construction.

b) One-class classification: This approach models the normal behavior of the application by learning patterns from positive instances. When a new instance arrives, this instance is classified by the model resulting in a score. This score, that is not necessary a probabilistic one, is compared to a threshold to determine whether this new request belongs to the model. This approach, which could also be called *Anomaly*, *Outlier* and *Normality* detection, as far as we know has never been applied to web application protection before.

c) Analyzing the whole HTTP *request:* In [28], Kruegel and Vigna propose an anomaly detection approach where the attributes of the model capture information about the parameters of the URI. They focus on characteristics like the parameter length, the order in which they appear and even generate a probabilistic grammar for each parameter. In both of our approaches we analyze the hole request information and not only the URI parameters. We think this provides further guarantees because all fields are subject to malformed input attacks, as illustrated in the ECML/PKDD2007 dataset [25].

d) Token abstraction: Several authors do not directly work on the tokens themselves, but rather on some simplification of them. In [29], Corona et al. abstract away numbers and alphanumeric sequences, representing each category with a single symbol. Torrano, Perez and Marañón [30] present an anomaly detection technique where instead of using the tokens themselves, they use a simplification that only considers the frequencies of three sets of symbols: characters, numbers and special symbol, as well as the list of special symbols used in each field. In general, what is counted is the number of symbols after applying some forget functor on the set of tokens, which abstracts away irrelevant differences. Removing information simplifies and speeds up the analysis by reducing the combinatory of possible tokens. It also reduces the risk of overfitting. However, if too much information is removed from the tokens some attacks may become indistinguishable from correct behavior, so an adequate compromise shall be found. In our experiments, the forget functor performs the following

[2]This is not surprising, as web applications arose during the 2000, and SQL injection was first reported in December 1998 [23]

transformations: (a) letters are transformed into lowercase and (b) accents are removed (for instance, the letter *á* is transformed into *a*). In addition to this, in the n-gram approach, digits are replaced by the capital letter "*N*". Such transformations provide a good balance, colapsing all the possible variants of the *select* keyword using in SQL injection attacks (namely, Select, SeLeCt, etc.) but still enabling to differentiate this keyword from other pieces of natural language text which may be part of text field in a normal request.

e) N-grams made of words: The contents of each request field is a string, so most of the anomaly detection approaches make use of document classification, NLP and information retrieval techniques. As content languages may significantly vary from one application to another (and even from one HTTP field to another), in most of the previous work tokens are just n-grams [11, 12, 28, 30, 31, 29, 5]. An n-gram is usually a sequence of consecutive characters, even though [31] shows that using n characters which are m positions away one from the other may also be an alternative. We generalize and extend the idea of n-grams and words in the following way: the contents string is first split into a sequence of sub-strings (tokens) separated by other sub-strings meeting a regular expression (the *delimiters*). Then, we consider n-grams formed with sequences of these tokens. This enables a more accurate analysis of some HTTP fields such as the URI, where there exists a clear delimiter character (the slash) which enables to divide the contents into tokens, namely, the words of the directory resources. Other fields, such as `Accept-Encoding` or `Accept-Language` also make use of commas and semicolons as delimiters, and can be addressed in terms of words instead of characters.

f) Fine granularity in the analysis: Among the works focusing on the application layer, a few of them consider each HTTP request as a single unstructured document to be analyzed and classified [26, 14]. This approach is prevalent when focusing on the current state of the art in attacks, and is also the approach proposed in the one-class classification technique presented in this paper. Other works perform some partial parsing of the HTTP structure, for example in [32, 28]. However, in this latter works, the parsing is reduced to the structure of the URI, splitting the URI into the URL and the web application parameters. In our proposal for n-gram analysis we go further, splitting the request contents into the contents of its header, header fields, and web application parameters (either appended to the URI or in the HTTP body). We advocate for this approach because each field has its own independent language, which may be quite different from the language of other fields. As a consequence, a much more specific and biased contents can be extracted from each one than from the whole request.

g) Higher order n-grams: Initial attempts for using n-gram analysis on HTTP requests focused on monograms and single character distributions [11, 30]. However, Kolesnikov and Lee [33] and Pastrana et al. [34] have shown that short n-grams are vulnerable to mimicry attacks, in which the attacker carefully adds characters in order to get closer to the n-gram distribution expected by the WAF. Mimicry attacks become much more challenging when using higher n-grams [12]. Opposite to the cited proposals, we do not use n-grams of a fixed length n, but all the n-grams of any length up to a given model parameter n that can be adjusted for each case. Moreover, in our proposal the n-gram length may be configured for each field. In this way it is possible to set $n = 3$ for all the model fields, but restrict the analysis to monograms on cookies, email addresses or other fields where higher order n-grams are too sparse but not all symbols are allowed.

VI. Conclusion and Further Work

We have presented two complementary approaches to enhance the detection capabilities of the ModSecurity WAF.

We follow a one-class classification approach to learn the behavior of normal web application requests based on features that represent the payload of an attack. Requests with abnormal behavior are classified as attacks. This classification process draws its model attributes from the knowledge that the security experts have concerning the current state of the art in web application attacks. The values of those attributes are then trained for nominal cases and learned from the normal behavior of the application. This classification approach provides the means for addressing attacks performed on fields which are expected to contain weakly biased data.

We also put forward an anomaly detection approach based on the expected n-gram frequencies for the fields and web parameters of the application. For that we make use of higher order n-grams of multiple lengths. We have experimented with several variants of this general framework and described results obtained with some particular instances of the model parameters. Anomaly detection draw its attributes from the expected normal behavior of the application, and deems as an attack those HTTP requests which deviates from such behavior. This positive characterization of the normality makes the WAF more resilient to zero-day attacks.

Experiments have been performed on three datasets of very different nature. The results on these datasets show the potential of machine learning approaches in the development of web application firewalls. The proposed method using a one-class classifier provide better detection and false positive rates than ModSecurity configured with the OWASP CRS out of the box. Regarding the n-gram approach, we show that in the two datasets used for testing it, the results clearly outperform the ones of ModSecurity. As we said before, the two approaches can be seen as complementary, the one-class can be used in Scenario I, when no specific training data is available, and the n-gram is suitable for Scenario II since it needs specific training data to learn from normal traffic.

A mid-term goal is to implement a module enabling to introduce machine learning techniques into ModSecurity itself. In the meantime, the prototypes implemented for these experiments can be immediately used to shorten the time required for tuning the Core Rule Set for a particular web application: running them on ModSecurity's logs and analyzing those HTTP requests that are deemed normal by the tools to spots false positive examples that can be used as basis to tune the rules.

Regarding the combination of one-class and ModSecurity, in this work we proposed a simple combination based

on the output labels of both components (normal, attack). In future work we will study the combination of the scores provided by both of them. These scores contain more information than just the classification outcome. In the area of classifier combination, there are several alternatives to improve the classification fusing the scores, or posterior probabilities, of multiple classification systems.

An alternative approach to address random contents fields is to apply a symmetry approach to anomaly detection, namely, use n-grams to learn the language signature of the malicious payloads for each attack technique in the current state of the art, and deem as normal the contents of any field that do not match such signature.

In order to push further the use of machine learning techniques in WAF technology, there is a strong need for disposing of real, open, free, standardized and documented datasets of several web application behavior. Such datasets shall offer a documented description of what web application functionalities are covered in the dataset, and the type of attack payloads that were used for testing. They shall also include erroneous or atypical input which does not correspond to attacks, in order to test the actual TNR that the ML technique offers. Should an anonymization procedure be required to publish the dataset, it shall respect the structure and correlation of original data. These goals can only be achieved by automated tools for building such datasets for a given application. Providing such tools is part of our further research agenda.

REFERENCES

[1] OWASP. Owasp top ten project. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] I. Trustwave Holdings, "Modsecurity: Open source web application firewall." [Online]. Available: http://www.modsecurity.org/

[3] OWASP. Owasp modsecurity core rule set project. [Online]. Available: https://www.owasp.org/index.php/

[4] C. Folini. (2016) Handling false positives with the owasp modsecurity core rule set. [Online]. Available: https://www.netnea.com/cms/apache-tutorial-8_handling-false-positives-modsecurity-core-rule-set/

[5] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A close look on n-grams in intrusion detection: Anomaly detection vs. classification," in *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, ser. AISec '13. New York, NY, USA: ACM, 2013, pp. 67–76.

[6] T. Berners-Lee, R. Fielding, and L. Masinter, "Rfc 2396: uniform resource identifiers (uri)," *IETF RFC*, 1998.

[7] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Irish conference on Artificial Intelligence and Cognitive Science*. Springer, 2009, pp. 188–197.

[8] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://www.jstor.org/stable/2984875

[10] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[11] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 203–222.

[12] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 226–248.

[13] A. P. V. Carmen Torrano Giménez and G. Á. Marañón, "CISC2010 dataset," http://www.isi.csic.es/dataset/, January 2012.

[14] "Analyzing web traffic: Ecml/pkdd 2007 discovery challenge," http://www.lirmm.fr/pkdd2007-challenge/.

[15] K. L. Ingham and H. Inoue, "Comparing anomaly detection techniques for http," in *RAID*, vol. 4637. Springer, 2007, pp. 42–62.

[16] OWASP. The owasp zed attack proxy. [Online]. Available: https://www.owasp.org/index.php/

[17] A. Riancho, "w3af-web application attack and audit framework," *World Wide Web electronic publication*, p. 21, 2011.

[18] C. Bockermann, M. Apel, and M. Meier, "Learning sql for database intrusion detection using context-sensitive modelling (extended abstract)." in *DIMVA*, ser. Lecture Notes in Computer Science, U. Flegel and D. Bruschi, Eds., vol. 5587. Springer, 2009, pp. 196–205.

[19] D. Kar, S. Panigrahi, and S. Sundararajan, "Sqlidds: SQL injection detection using document similarity measure," *Journal of Computer Security*, vol. 24, no. 4, pp. 507–539, 2016.

[20] ——, "Sqligot: Detecting SQL injection attacks using graph of tokens and SVM," *Computers & Security*, vol. 60, pp. 206–225, 2016.

[21] M. Quadrant, "Magic quadrant for web application firewalls," *Analyst (s)*, p. G00314552, 2017.

[22] L. L. MIT, "Darpa intrusion detection evaluation - intrusion detection attacks database," https://www.ll.mit.edu/ideval/docs/attackDB.html, January 2012.

[23] J. F. (signing as rain.forest.puppy), "Nt web technology vulnerabilities," *Phrack Magazine*, vol. 8, no. 54, December 1998.

[24] M. Exbrayat, "Ecml/pkdd challenge: analyzing web traffic a boundaries signature approach," 2007, p. 53.

[25] K. Pachopoulos, D. Valsamou, D. Mavroeidis, and M. Vazirgiannis, "Feature extraction from web traffic data for the application of data mining algorithms in attack identification." Citeseer, 2007.

[26] B. Gallagher and T. Eliassi-Rad, "Classification of http attacks: a study on the ecml/pkdd 2007 discovery challenge," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., July 2009.

[27] C. Raıssi, J. Brissaud, G. Dray, P. Poncelet, M. Roche, and M. Teisseire, "Web analyzing traffic challenge: description and results," in *Proceedings of the ECML/PKDD*, 2007, pp. 47–52.

[28] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM, 2003, pp. 251–261.

[29] I. Corona, D. Ariu, and G. Giacinto, "Hmm-web: A framework for the detection of attacks against web applications," in *Proceedings of IEEE International Conference on Communications, ICC 2009, Dresden, Germany, 14-18 June 2009*, 2009, pp. 1–6.

[30] C. Torrano-Gimenez, A. Perez-Villegas, G. Á. Marañón *et al.*, "An anomaly-based approach for intrusion detection in web traffic," *Journal of Information Assurance and Security*, vol. 5, no. 4, pp. 446–454, 2010.

[31] D. Ariu, I. Corona, G. Giacinto, and F. Roli, "Mcpad and hmmweb: two different approaches for the detection of attacks against web applications," in *Italian Workshop on Privacy and Security (PRISE)*, Rome, 2008.

[32] J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo, "Measuring normality in http traffic for anomaly-based intrusion detection," *Computer Networks*, vol. 45, no. 2, pp. 175 – 193, 2004.

[33] O. Kolesnikov and W. Lee, "Advanced polymorphic worms: Evading ids by blending in with normal traffic," Georgia Tech, Tech. Rep., 2004.

[34] S. Pastrana, C. Torrano-Gimenez, H. T. Nguyen, and A. Orfila, *Anomalous Web Payload Detection: Evaluating the Resilience of 1-Grams Based Classifiers*. Springer International Publishing, 2015, pp. 195–200.