An IDS for Web Applications

A. Biscotti, G. Capuzzi, E. Cardinale, L. Spalazzi
DIIGA — Università Politecnica delle Marche — I-60131 Ancona - Italy
e-mail: a.biscotti@univpm.it, {capuzzi, cardinale, spalazzi}@diiga.univpm.it

***Abstract*—This work presents a WEB-IDS that combine both anomaly and misuse detection approach. This mixed solution is really interesting because merges the two complementary methods used to recognize attacks; we solved the usual conflicts presented by this choice and obtained an higher results accuracy. Our tool starts with the misuse-based module and its results are passed to the anomaly detection module: in this way the system has an high reactivity, less false negatives, it is simplier to solve conflicts between the two modules and the anomaly based module do not need to process dangerous events recognised by the first module. Our system does not need any specific setting, but only a training period. There are also different auto-setting tresholds for the different resources that reduce false alarms. The system is implemented as system service and tested with a real dataset by a services company.**

I. Introduction

Web servers and web applications are often under attack by malicious software or intruders. The most part of web applications or web server-extentions are not structured with a secure criteria, so the number of attacks to them is increasing. Intrusion Detection Systems are provided with signatures to reveal attacks to this kind of applications; unfortunately is really difficult to mantaine updated the signatures because the high number of new vulnerability daily discovered. To avoid this problem, this kind of IDS shoud be complemented by anomaly detection systems that consent to discover attacks with unknow signature. In literature the possibility to combine misuse and anomaly detection was analyzed by E. Tombini, *et al.* [6], evidencing that in web application is better to have first the misuse module and then the anomaly one. The concept of anomaly detection was proposed for the first time by D.E. Denning [4], that presented an abstract model of a real-time intrusion detection system (IDES), based on the convinction that an use of the system different from the previous uses may be a signal of an improper use. A lot of other techniques were proposed to approach the anomaly intrusion detection by A.K. Ghosh, *et al.* [9] that used a neural network model to identify anomalous beaviors; L. Portnoy, *et al.* [12] proposed an unsupervisioned clustering algorithm to classify normal and abnormal activity; T. Lane and C.E. Brodley [13] presented a machine learning model based on IBL (instance-based learning) applyied to system applications. H. Feng, *et al.* [7] improved the static analysis for intrusion detection using PDA (push-down automaton). The limits of the state of the art is that all this works refers to system programs that are static: their beavior doesn't change to much working so is simply to identify anomalies in their working. In the case of web applications is different: we have dinamic entities and their behavior is strictly dependent from the interaction with the users; for this reason are interesting statistical and probability techniques proposed in their works about intrusion detection on web applications by C.Kruegel, *et al.* [15]. Cisco Systems [2] or Real Secure [11], implemented in their commercial systems solutions that combine anomaly and misuse detection, but they are principally misuse based. Therefore the target of this project is to improve web server and web applications security with many innovative algorithm modifications respect to common IPS, obtaining better performance in discovering attacks that exploit specific vulnerabilities for that applications, especially those developed in-house by service companies.With this paper we analyzed a possible combination of two different detection modules (misuse-based and anomaly-based) and our target was to determinate the capability and the efficiency of this combination to discover web-based attacks. Our attention is focused to obtain the less critical automatic solution of the conflicts between the two different evaluations on a same security event too, by developing an update engine which automatically adjust the sensitivity of the system to reduce the number of false alarms. Another objective was to study the possibility to add an efficient automatic prevention engine in order to decrease the quantity of critical data to manage for the system administrator. The basic problem of this solution is that web applications developed in-house by service companies does not provide a list of signatures, so it is necessary a certain time before the system can learn the attributes of applications and either a complete list of intrusive events or profiles of normal utilization of these applications have to be created, before the system achieves the best efficiency.

The paper is structured as follows: Section II provides an overview of the Detection Model; Section III, describes the Misuse detection Model; Section IV presents the evaluation tests of the system; Section V describes conclusions.

II. Overall Detection Model

Our system analyzes on-line a series of temporal subsequent HTTP requests as logged by most common web server (for example, Apache [8]). The analysis focuses on either GET or POST requests to HTML pages, server-side programs or active documents and consists of a serial combination of misuse and anomaly detection, with misuse detection first. More formally, the input to the process is a request R, extracted from web server access log file. A request R can be expressed in several ways depending of custom web server logging directives. In our work, we assumed that requests were logged in the Common Log Format (CLF) or Extended Common Log Format (ECLF). In these formats the most important features logged by web server are the IP source address, the date and the time of the request, the path to the desired web page

Proceedings of the 2008 High Performance
Computing & Simulation Conference ©ECMS
Waleed W. Smari (Ed.)
ISBN: 978-0-9553018-7-2 / ISBN: 978-0-9553018-6-5 (CD)

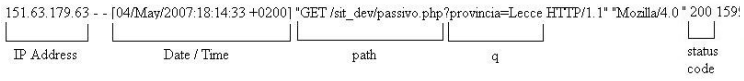

Fig. 1. Sample web server access log entry

(*path*), and sometimes an optional query string (*q*). The query string is used to eventually pass parameters to the referenced script and it is identified by a leading "?" character. A generic query string normally consists of an ordered list of *n* pairs of parameters with their corresponding values, so q can be expressed as $q = (p_1, v_1), (p_2, v_2), ..., (p_n, v_n)$ where $p_i \in P$, the set of all parameters and v_i is a string. Another important feature is the status code of the request. Figure 1 shows an example of a request logged by web server in ECLF format, in which elements used in the analysis are underlined. Let E be the event processed by the system (in this case the last line of the access log file). Additionally let M_i and M_u be the subsets of E that represent the sets of events declared respectively *intrusive* or *unknown* by the misuse detection component and let A_s and A_u be the subsets of E that represent the sets of events declared respectively *safe* or *unsafe* by the anomaly detection component. In this combination the misuse component tries to match E with a set of signatures and raises an alarm if the result of pattern matching is true (M_i). Events declared unknown in the first step (M_u) are then analyzed by an anomaly detection component. If the anomaly detector declares E safe, the event is assumed completely irrelevant (A_u) and is filtered out. On the contrary, an alarm is raised if E is declared intrusive (A_i). The detection process is shown in Figure 2. Events in the subset $M_u \cap A_u$ represent the set of events declared unsafe (potentially intrusive) by the anomaly detector and unknown (potentially safe) by the misuse detector. These events cannot be interpreted automatically because they should be considered as false positives issued by the anomaly detector or false negatives issued by misuse detector and the anomaly model (respectively the signatures database) must be updated. In this case the main task of system administrator is signaling what component gave wrong response, and an update engine automatically provides to update the anomaly detection component or add a new signature to the misuse detection component. At this time, in the first case the event tagged as normal is used to refine the anomaly model by computing a new anomaly threshold for the analyzed script, as shown more accurately in section about Anomaly detection Model. In the second case the automatic generation of a signature is made simply adding the tagged request at the end of the list of regular expressions. However in some cases, the system tries to automatically apply some prevention rules in order to block the communication with client who generated the anomaly event, with the help of a firewall. The prevention model is described in section Prevention Model.

III. Misuse Detection Models

Misuse detection Model: The misuse detection component is used to detect attacks embedded in URLs and report known malicious requests. For example, the presence in the request of the words "SELECT" or "WHERE" could be used to detect an attempted incorrect use of SQL commands to retrieve

Fig. 2. System architecture

sensible data. The misuse detection process uses a list of regular expressions, which items represent manifestations of the most common attacks. These regular expression are used to match against the request analyzed and an alarm is raised when a match occurs. We used the Java regex package to implement the mechanism of pattern matching, which is in conformance with Level 1 of Unicode Technical Standard #18: Unicode Regular Expression Guidelines [3], plus RL2.1 Canonical Equivalents. Currently, the list contains about 70 regular expressions, but new attacks can be detected by adding new regular expressions. A new expression can be added manually without stopping system execution or it can be automatically generated and added by the system when a suspicious request is tagged as intrusive by system administrator. Regular expressions are compiled at runtime, before the matching process starts. The list of regular expressions is specified into a text file, in an XML-based format. An example is shown below.

```
<signatures>
  <expression name="Cross-site Scripting">.*(script)+.*
  </expression>
  <expression name="Directory Trasversal">.*(\.\./)+.*
  </expression>
<expression name "SQL Injection">.*
                (select|insert|update|delete|union|--)+.*
</expression>
[...]
</signatures>
```

At this time, the automatic generation of a signature is made simply adding the tagged request at the end of the list of regular expressions. *Anomaly detection Model*:the anomaly detection component provides a model of the "normal behavior" of users and applications. The basic assumption is that, in case of intrusive actions, the behavior of users or applications differs substantially from normal behavior and this difference can be expressed quantitatively. The behavior of software entities in the web applications context is very dinamic and it is strictly connected with user interaction, which often governs the visualization process of data and informations through the choice of a set of values associated with some parameters. Additionally each web application is different from others in terms of number and type of parameters to elaborate. Our approach was initially based on the model proposed by C.Kruegel, *et al.* [15], which uses several different *evaluations* about requests, parameters and their relationship to detect anomalous entries. An evaluation is a set of statistical procedures used to evaluate a certain *feature* of a request. A feature can be related to a single parameter of a query string (e.g. the string length of a particular parameter value), to all parameters (e.g. the order of parameters in a query string), or to some relationship between a request and others related to a specific web page or script *r* (e.g. number of requests in a

time slot). The model operates in two distinct steps: in a first step (*learning* or *tuning*) each evaluation is performed on a sufficiently large set of normal requests relating to the same web page or script (first 1000 in our implementation)and on their parameters in order to build a *profile* for each web page or script;for this purpose we used historical access log data files, gathered from the system administrator of a small Italian company . Afterward, a detection threshold is estabilished by evaluating requests separately. In the detection phase, the task of these evaluations is to assign a probability value to either a request as a whole or one of its parameters. This probability value reflects the probability of the occurrence of the given feature value with regards to an established profile. The assumption is that feature values with a sufficiently low probability can indicate a potential intrusive behavior. Based on the evaluation outputs, a request is either reported as a potential intrusive behavior or as normal. This decision is reached by calculating an *anomaly score*. A request is reported as anomalous if this anomaly score is above the corresponding detection threshold. The anomaly score value is calculated using a weighted sum as shown in Eq.(1). In this equation, w_e represents the weight associated with evaluation *e*, while p_e is its returned probability value. The probability p_e is subtracted from 1 because a value close to zero indicates a possible anomalous event. In this case it should yield a relative high anomaly score. The w_e values are estabilished a priori and they can be adjusted regarding the analyzed application, web page or script, after a brief analysis process on historical data of web server.

$$AS = \sum_{e \in Evaluations} w_e * (1 - p_e) \tag{1}$$

Our evaluation procedures are very similar to those proposed by Kruegel et al., but we used only six anomaly evaluations to perform detection process and we refined evaluation procedures in some points. The evaluations used are enumerated below:

Length of values associated with parameters;
Distribution of characters in values associated with parameters;
Presence of limited set of values associated with parameters;
Presence or absence of a parameter in a request;
Order of parameters in a request;
Access frequency to a web page or script;

When a certain request contains several parameters, we chose to consider the lower probability value returned by our anomaly evaluations, in order to avoid a malicious user to hide a single invalid input into a series of valid inputs. Additionally, when a script without parameters to elaborate is analyzed, only two evaluations are performed: the check of presence or absence of parameters in the request and the access frequency to the script.

Length of values associated with parameters:The length of a value associated with a parameter of a request can be used to detect anomalous requests, especially when these values are either fixed-size tokens or short strings derived from user input (such as fields in a form). For example, to overflow a buffer in a target application, it is necessary to send a large amount of data, depending on the length of the buffer. In the training phase the goal of this evaluation is to approximate the actual but unknown distribution of the parameter lengths, so in the detection phase it can detect instances that significantly deviate from the observed normal behavior. Clearly, the probability density function of the underlying real distribution often doesn't follow a smooth curve. Additionally the distribution has a large variance in several cases. Nevertheless, the evaluation is able to efficiently identify significant deviations.

LEARNING:in the learning phase, the length of values associated with a certain parameter in a request can be expressed as a random variable, and it is possible to calculate mean μ and variance σ^2 associated with it. The mean and the variance of the real parameter values length distribution are so approximated by calculating the sample mean and the sample variance for the lengths $l_1, l_2, \ldots, l_n$ of the values associated with the analyzed parameter and processed during the learning phase (assuming that *n* requests with that parameter were processed). The cost of this evaluation is proportional to the number *n* of queries analyzed during the learning phase, followed by a constant cost to compute the mean and the variance.

DETECTION:In the detection phase we already know the estimated parameter length distribution, in particular μ and σ^2. The task of this phase is to evaluate the anomaly of a value associated with the analyzed parameter with length *l*. We assume that a value with length less than μ, relating to a specific parameter, is associated with a probability value of 1. This is due the fact we consider this evaluation able to detect anomalous requests in which a large amount of data is injected in one or some values. To evaluate the anomaly of a string with length *l* higher than μ, we quantify the "distance" of the length l from the mean value μ with the help of the Cantelli inequality [5]. The Cantelli inequality is an efficient metric to model decreasing probabilities for strings with lengths that increasingly exceed the mean. It puts, for an arbitrary distribution with mean μ and variance σ^2, an upper bound on the probability that a generic length *x* is higher than *l*, as shown in Equation 2. When *l* is very distant from μ then the probability value associated with a string having a greater length than *l* should decrease. In this case an attacker cannot insert malicious input by padding the string and increasing its length, because an increase in length reduce the probability value associated with that string.

$$p(l) = \begin{cases} 1 & l \leq \mu \\ \frac{\sigma^2}{\sigma^2 + (l-\mu)^2} & l > \mu \end{cases} \tag{2}$$

Distribution of characters in values associated with parameters When we analyze a request, sometimes we are able to detect an anomalous value associated with a certain parameter by looking at its distribution of characters. This is due the fact that often user input have a regular structure, is mostly human-readable, and almost always contain only printable characters. Additionally, a large percentage of characters in regular values are drawn mainly from letters, numbers, and sometimes a

few special characters. However, there are some cases in which a malicious user sends binary data (e.g., buffer overflow attacks), or inject a string with a noticeable strange character sequence (e.g., with many repetition of the dot character in directory traversal exploits). Anyway, when we analyze a set of several requests containing same parameters, similarities often can be observed between the character frequencies of values associated with a certain parameter. In fact, if we consider a string drawn from Italian or English language, we observe that there are words in which some characters are more frequent than others but often there is no one that is clearly more prevalent than others. So, if we sort in descending order the relative frequencies for all possible characters in a legitimate string, one can expect that the relative frequencies slowly decrease in value. In case of malicious input, instead, these frequencies often drop extremely fast. This can be the result of a certain padding character that is repeated many times in a buffer overflow attack or because of the many occurrences of the dot character in a directory traversal attempt. Therefore we assume that the character distribution of a value is represented by the array of its relative frequencies sorted in descending order.

LEARNING the goal of the learning phase is to build a profile of the normal character distribution of a generic value associated with a certain parameter, for each parameter of a script. For this purpose, the character distribution for each observed parameter is stored. Then an "idealized" character distribution is approximated by setting, for each index *i* of the relative frequencies array, the average of all *n* values of stored character distributions that are in the position *i*, assuming that *n* requests with that parameter are analyzed. Because all individual character distributions sum up to unity, their average will do so as well, and the idealized character distribution is well-defined. The cost of building this model is linear in the number of parameters that are analyzed during this phase. For each parameter, the character distribution has to be determined, an operation which has a cost that is proportional to the length of its value string.

DETECTION: the task of the detection phase is to determine the probability that the character distribution of a certain value is conforming with the modelized legitimate character distribution associated with its reference parameter. In this case, more precisely, we want to check if this array is a sample drawn from a population with a certain distribution, that can be represented by the idealized character distribution of its reference parameter. This probability, or more precisely, the confidence in this hypothesis is calculated by the Pearson χ^2 statistical test[5]. We chose to group the values of the array in six intervals defined as follows:[0], [1, 3], [4, 6], [7, 11], [12, 15], [16, 255], assuming that possible characters are drawn from a subset of 256 character (ASCII - 8 bit) and reflecting the fact that the relative frequencies are sorted in descending order, so values in position *i* are higher when *i* is small. Additionally, the value representative of an interval is the average of values between the considered indexes. When a new parameter is analyzed, the number of occurrences of each character in the string value is determined. Afterward, the values are sorted in descending order and combined by aggregating values that belong to the same interval *i*. The resultant value is indicated as O_i. The χ^2 value is so determined as shown in Equation 3. In this equation E_i represents the expected value and it is calculated by multiplying the value of the interval *i* for the length of the string analyzed.

$$\chi^2 = \sum_{i=0}^{i<6} \frac{O_i - E_i}{E_i} \tag{3}$$

The χ^2 test is then used to calculate the probability that the given sample has been drawn from the legitimate character distribution. The actual probability *p* is read from a predefined table (Table I) using the χ^2 value and the degrees of freedom as index, observing that in this test the degrees of freedom are calculated as $(numberofintervals - 1)$. The derived value *p* is used as the return value for this model. When the probability that the sample is drawn from the legitimate character distribution increases, *p* increases as well. The cost of evaluate an input string using this method consists of the calculation of the χ^2 value[].

TABLE I
PROBABILITY VALUES FOR 5 DEGREES OF FREEDOM

χ^2	*p*	χ^2	*p*
0.41	0.995	9.24	0.1
0.55	0.99	11.07	0.05
0.83	0.975	12.83	0.025
1.15	0.95	15.09	0.01
1.61	0.9	16.75	0.005

Presence of limited set of values associated with parameters:the purpose of this evaluation is to determine whether the values of a certain parameter are drawn from a limited set of possible alternatives (i.e., "checkbox", "radio" or "select" HTML input types). When a malicious user attempts to use these parameters to pass to the analyzed script illegal values or values that are not in the trusted set, the intrusion attempt can be detected. When a set of possible alternatives can be identified for a certain parameter, it is assumed that the parameter values are random and no attacks can be detected by this evaluation.

LEARNING:the goal of this phase is to model when the values associated with a certain parameter are random or part of an enumeration. When a set of different occurrences of the analyzed parameter values are observed, one can see that in the case of an enumeration, the number of different values encountered does not exceed a certain unknown threshold *t*. Instead, when the number of different parameter values grows proportional to the total number of analyzed instances, the use of random values is indicated. It is consequently necessary to estabilish the correlation between the total number of analyzed values and the total number of different values encountered during this phase. More formally, to decide if a parameter is associated with an enumeration, we calculate the statistical correlation ρ between the values of the functions f and g, defined $\forall x \in \{1, 2, ..., n\}$ as follows on N_0, where *n* is the total number of values encountered during this phase, and $S^{(x)}$ represents the set of values encountered at the time the x^{th}

value is analyzed:

$$f(x) = x \tag{4}$$

$$g(x) = \begin{cases} g(x-1)+1 & x^{th} value \in S^{(x)} \\ g(x-1)-1 & x^{th} value \notin S^{(x)} \\ 0 & x = 0 \end{cases} \tag{5}$$

The correlation parameter ρ is derived after the training data has been processed. It is calculated from *f* and *g* with their respective variances Var(*f*), Var(*g*) and the covariance Covar(*f*,*g*) as shown below:

$$\rho = \frac{Covar(f,g)}{\sqrt{Var(f) * Var(g)}} \tag{6}$$

If ρ is less than 0, then *f* and *g* are negatively correlated and an enumeration is assumed, reflecting the fact that increasing the value of observed values, the number of different occurrences has not shown a proportional increase too. This means that some values was encountered several times during the training phase. In the opposite case, where ρ is greater than 0, the values observed have shown sufficient variation to support the hypothesis that they are not drawn from a small set of predefined values. Naturally, when an enumeration is assumed, the complete set of values encountered is stored for use in the detection phase. The cost of building this model is strictly connected to the calculation of the covariance between these two simple functions. This cost depends on the number of analyzed requests.

DETECTION:once it has been determined that the values of a certain parameter of a script are tokens drawn from an enumeration, any new value *v* is expected to appear in the set *S* of known values. When this happens, a probability value of 1 is returned. If the value is not in the established set of values, a probability value of 0 is returned, as shown in Equation 7. If it has been determined that the parameter values are random, the model always returns 1. In order to increase the efficiency of the detection we use the Java HashMap data type to store and retrieve values.

$$p(v) = \begin{cases} 1 & v \in S \\ 0 & v \notin S \end{cases} \tag{7}$$

Presence or absence of a parameter in a request: Most of the time, server-side programs or scripts are not directly invoked by users typing the input parameters into the URIs themselves. Instead, client-side programs or scripts pre-process the data and transform it into a suitable request. This usually results in a high regularity in the number, name, and order of parameters. Another interesting situation is when developers with little expertise use *"hidden"* type forms in their HTML pages and then process the data in a separate script. In this case this type of parameters does not appear in requests logged by the web server, so its presence can indicate the attempt of an intrusion. The analysis performed by this evaluation takes advantage of these facts and tries to detect requests that deviate from the estabilished profile, built in the training phase. This evaluation described in this section, deals with the presence and absence of parameters p_i in a query string *q* and consider a query as a whole, differing from previous ones, which focus on features of individual query parameters. This approach assumes that the absence or abnormal presence of one or more parameters in a query might indicate malicious behavior. This allows for the detection of intrusion attempts where server-side applications or scripts are probed or exploited by sending malformed requests.

LEARNING: the goal of this phase is to estabilish when a script is associated with some parameters and, in the positive case, to create a model of acceptable subsets of parameters that appear simultaneously in a query string. This is done simply by storing each distinct subset $S_q = \{p_i, ..., p_k\}$ of parameters seen during the training phase.

DETECTION: the detection phase have to deal with three different situations: *Scripts without visible parameters* - In this case the evaluation is performed observing if the analyzed request has one or more parameters in its URI. In this case a probability value of 0 is returned, 1 otherwise; *Scripts with visible parameters* - In this case, for each request analyzed, the current parameter set is extracted. When the observed set of parameters has been encountered during the training phase, 1 is returned, otherwise 0. *Generic scripts* - This situation deals with scripts that can appear with or without parameters. In this case the detection process is performed with the method described in the situation 2, when a set of parameters can be extracted from the analyzed request. Otherwise a probability value of 1 is returned. The current script type is determined by reading a code of two boolean values, set in the training phase, where *11* is a generic script, *10* is a script without visible parameters, *01* is a script with visible parameters and *00* is for the Unknown.

Order of parameters in a request:As discussed in the previous section, legitimate requests often contain the same parameters in the same order. This is usually not the case for hand-crafted requests, as the order chosen by a malicious user can be arbitrary and has no influence on the execution of the program. The goal of this evaluation is to determine whether the given order of parameters is consistent with a profile built during the learning phase.

LEARNING The order constraints between all *k* parameters of a legitimate query string are determined during this phase. It is consequently necessary to estabilish an order relationship between parameters. So we assume that a parameter p_t of a script precedes another parameter p_s when p_t and p_s appear together in the parameter subset of at least one query string and p_t comes before p_s in the ordered list of parameters of all queries where they appear together. In order to store these relationships between parameters we use a matrix *M* of (*s* x *s*) elements, where *s* is the total number of parameters associated with the analyzed script and $M_{ij} = 1$ if p_i precedes p_j, 0 otherwise. So, for every query string q_i, with *i* = 1,. . . ,*n*, that is analyzed during the training period, the ordered list of its parameters $p_1, p_2, ..., p_i$ is processed. For each attribute pair (p_t, p_s) in this list, with $t \neq s$, the M_{ts} value is set to 1. In this phase all analyzed requests are assumed to be normal, so the final result is that $M_{ts} \neq M_{st}$ when p_t precedes p_s, assuming that parameters can only appear in the same order. If parameters p_t and p_s are admitted in both different orders, then $M_{ts} = M_{st}$. The cost of building this profile is not high, because the total number of parameters processed by a generic

script is usually relatively small.

DETECTION The detection process checks whether the parameters of a query string satisfy the order constraints determined during the learning phase. Given a query string with parameters $p_1, p_2, ..., p_i$ and the matrix *M*, all the parameter pairs (p_j, p_k), with $j \neq k$, are analyzed to detect potential violations. A violation occurs when for any single pair (p_j, p_k) of the current query string, where p_j precedes p_k, the corrisponding M_{jk} value is 0. In this case the evaluation returns a probability value of 0, otherwise it returns 1.

Access frequency to a web page or script:different server-side applications or web scripts normally are invoked with different frequencies. However, after monitoring a specific web page or script in a sufficiently long time interval, one can often observe that the general access patterns remain relatively constant. It is possible to distinguish between two types of access frequencies for each web page or script. One is the frequency of the application being accessed from a certain client (based on the IP address), the other is the total frequency of all accesses. When a malicious user attempts a DoS (Denial of Service) exploit for a certain script, the number of accesses observed in a small time interval from that client can increase drastically. Otherwise the frequency of all accesses can grow extremely in case of DDos (Distributed DoS) attempts. Changes in access patterns can indicate intrusion attempts (e.g., when an application is usually accessed infrequently but is suddenly exposed to a burst of invocations). This increase could be also the result of an attacker probing for vulnerabilities or trying to guess parameter values. A single determined attacker can evade detection by executing his actions slowly, but often most intruders use tools that execute brute force attacks, raising the total access frequency to a suspicious level.

LEARNING:the objective of this phase is to build a model of normal access frequency pattern for a web page or script. To determine the expected normal access frequencies, the time period between the first and the last request in the training data set is divided into consecutive time intervals of a fixed size (60 seconds in our implementation). Then, the total number of requests and the numbers of requests from distinct clients (distinct IP addresses) are counted in each of these intervals. The counts for the total accesses and the counts for the accesses from distinct clients can be considered as two random variables, which respective means and variances can be evaluated. These values can represent the normal web page or script behavior in terms of number of requests made to it. The cost of building this profile is proportional to the number of requests that are analyzed during the training period.

DETECTION:this evaluation focuses on whole sequences of queries, so it is necessary to maintain data of recent accesses to the analyzed script for a certain time interval. The main goal is to be able to detect vulnerability probing, parameter value guessing and DoS attempts. So during detection, time is divided into intervals of the same fixed size that was used during the learning phase. When a request is evaluated, the number of total requests n_1 and the number of requests from this client n_2, both in the current time interval, are determined. Similar to the attribute length evaluation, the Cantelli inequality is used to calculate the probability of n_1, given the mean and the variance of the total access frequencies, and the probability of n_2, given the mean and the variance of access frequencies from distinct clients. This two probabilities are then combined in a weighted sum, as shown below, and returned by this evaluation.

$$p(a) = \sum_{i=1}^{2} w_i * p(n_i) \tag{8}$$

The w_i values are initially set to $1/i$, but they can adjusted by the system administrator taking in account the specific web page or script type. The detection cost is proportional to the number of requests analyzed during the current detection interval. This module can be linked to a prevention module: we did it, but we do not talk about it in this paper.

IV. EVALUATION

This section describes the approach used in order to evaluate the intrusion detection system proposed. The evaluation was performed by gathering on-line (in two different experiment) real data from the main web server of an Italian company, SIMobile s.r.l.(www.simobile.it). The IDS was configured and integrated with the web architecture (e.g., operating system, web server, DBMS), by installing it as a service.

TUNING PHASE: the system tuning phase was performed off-line, analyzing a stored database of historical legitimate HTTP/HTTPS requests. So we usedthem to model the normal system behavior. Table II, shows relevant informations about the learning set, like data gathering time interval, total number of analyzed requests and total number of pages or scripts modeled.

DETECTION:there are several indexes to evaluate the effectiveness of an intrusion detection system. In our work, we used a set of typical indexes often used in diagnostic tests, as shown below.

TABLE II
LEARNING PHASE INFORMATIONS

Time interval	Log size	# of requests	# of modeled scripts
165 days	16 MByte	87284	260

Sensitivity (Sen)= $\frac{TP}{TP+FN}$,
Specificity (Spe)= $\frac{TN}{TN+FP}$,
Positive predictive value (PPV) = $\frac{TP}{TP+FP}$,
Negative predictive value (NPV) = $\frac{TN}{TN+FN}$,
Prevalence (Prev)= $\frac{TP+FN}{TP+FN+FP+TN}$,

Where TP,FN,FP,FN are respectively: true positive, false negative,false positive and false negative. The test has an high degree of effectiveness when sensitivity and specificity values are close to 1. Other two fundamental indexes are related to false alarms, as shown below.

False Positive Rate (FPR) = $\frac{FP}{TN+FP} = 1 - specificity$,
False Negative Rate (FNR) = $\frac{FN}{FN+TP} = 1 - sensitivity$.

Obviously, the test has an high degree of effectiveness when the false positive/negative rate is close to 0. The system has been evaluated by analyzing on-line the HTTP/HTTPS traffic towards the monitored web server. In our experiment the

weight w_e of Equation (1) has been set to $1/6 = 0,167$ and the weight w_i of Equation (8) has been set to $1/2 = 0,5$. Table III reports the results of one of the two experiments.

TABLE III
EVALUATIONS RESULTS

Monitoring days	55	alerts	28	Sen	1
Request logged	18894	TP	17	Spe	0,997
Suspicious events	28	TN	3655	FPR	0,003
Intrusive events	17	FP	11	FNR	0
		FN	0	PPV	0,607
		Prev	0,0046	NPV	1

We have a false negative when the system fails to identify a potentially intrusive behavior; therefore, we can not compute the number of false negatives with an automatic procedure. The number reported in Table III has been manually computed by the system administrator day by day. His task consisted in checking daily requests (around 343 requests per day) and identifying potential intrusions not signaled by the system.
The reliability of the evaluation depends on two factors: the total number of analyzed requests and the comparison with the related works. The total number of analyzed request is related to the average server traffic load of the monitored company, but however the system shows that results are comparable with the main reference work [15]. Anyway we expect a sensible improvement in reducing false positives when automatically updating the misuse engine with new specific signatures and the anomaly engine with adjusted thresholds after a longer monitoring period.

V. CONCLUSIONS

In this work we considered the security problem of web-applications and the application of Intrusion Detection Systems to this kind of systems.We proposed an intrusion detection model that improves the previous model proposed by [15]. I this model we combined an anomaly detection approach with a misuse detection approach. Indeed, the best way to reveal web application attacks is to use the precision of signature based systems with the flexibility of anomaly detection systems and to solve problems coming from the combination of two approaches. About anomaly detection we obtained a great advantage combining different evaluation systems to cover the great number of attack typologies. The model proposed doesn't need any specific configuration, but only a training period. We implemented this model in a IDS that we experimented in a real context.

TABLE IV
IPS COMPARISON

Sistem	Sensitivity	FP/ # logs	DATA
Our IDS	100%	0,02% - 0,06 %	Real Data
[15]	100%	0,002% - 1,45%	Simulated
[14]	-	0,069%	Real Data

Table IV shows that our results are comparable with the main reference work [15] in terms of false positive , with the difference that while their results came from a simulation, we applied our IDS to a real company network. Furthermore, the Positive Predictive Value is about 60 %, that is a tipical value of PPV for IDS/IPS, as described in [10] and [1]. False positives in particular, occured in the first days of monitoring and then decreased, as consequence of adjusting detection thresholds.

REFERENCES

[1] J.S. Baras. A.Cardenas and K. Seamon. A framework for the evaluation of intrusion detection systems. In *IEEE Symposium on Security and Privacy*.

[2] Cisco. Cisco intrusion prevention system. Technical report, http://www.cisco.com/en/US/products/sw/secursw/ps2113/index.html.

[3] Mark Davis. Unicode technical standard 18, unicode regular expressions. Technical report, http://www.unicode.org/unicode/reports/tr18.

[4] D.E. Denning. An intrusion detection model. volume 2, pages 222–232, 1987.

[5] Luc Devroye. In *Non-Uniform Random Variate Generation*, 1986. Springer-Verlag, New York.

[6] L. Me E. Tombini, H. Debar and M. Ducasse. A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic. December 2004.

[7] H. Feng, J. Giffin, Y. Huang, S. Jha, W. Lee, and B. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA,*, 2004.

[8] Apache Software Foundations. Apache http server log files. Technical report, http://httpd.apache.org/docs/2.2/logs.html.

[9] A.K. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. volume AZ, pages 259–267, December 1998.

[10] David Dagon Wanke Lee Guofei Gu, Prahlad Fogla. In *Measuring Intrusion Detection Capability: An Informmation-Theoretic Approac.*, March 2006. In Proceedings of ACM Symposium of InformAction, Computer and Communications Security(ASIACCS06).

[11] IBM. Ibm internet security sytems proventia network intrusion preventionsystem. Technical report, http://www.iss.net/products/product_ sections/Intrusion_ Prevention.html.

[12] S. Stolfo L. Portnoy, E. Eskin. Intrusion detection with unlabeled data using clustering. Novembre 2001.

[13] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction foranomaly detection. pages 150–158. ACM Press, 1998.

[14] M. Dacier M. Almgren, H. Debar. A lightweight tool for detecting web server attacks. In *ISOC Symposium on Network and Distributed Systems Security*.

[15] W. Robertson, G. Vigna, C. Kruegel, and R. Kemmerer. Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. In *Proceeding of the Network and Distributed System Security (NDSS) Symposium San Diego, CA*, 2006.