An Anomaly-Based Approach for Intrusion Detection in Web Traffic

Carmen Torrano-Gimenez[1], Alejandro Perez-Villegas[1], and Gonzalo Alvarez[1]

[1]Instituto de Física Aplicada, Consejo Superior de Investigaciones Científicas,
Serrano 144 - 28006, Madrid, Spain
{carmen.torrano, alejandro.perez, gonzalo}@iec.csic.es

***Abstract*:** A new system for web attack detection is presented. It follows the anomaly-based approach, therefore known and unknown attacks can be detected. The system relies on a XML file to classify the incoming requests as normal or anomalous. The XML file, which is built from only normal traffic, contains a description of the normal behavior of the target web application statistically characterized. Any request which deviates from the normal behavior is considered an attack. The system has been applied to protect a real web application. An increasing number of training requests have been used to train the system. Experiments show that when the XML file has enough information to closely characterize the normal behavior of the target web application, a very high detection rate is reached while the false alarm rate remains very low.

***Keywords*:** web attacks, intrusion detection system, anomaly intrusion detection, web application firewall, web application security.

1. Introduction

Web applications are becoming increasingly popular and complex in all sorts of environments, ranging from e-commerce applications to banking. As a consequence, web applications are subject to all sort of attacks. Additionally, web applications handle large amounts of sensitive data, which makes web applications even more attractive for malicious users. The consequences of many attacks might be devastating [1], like identity supplanting, sensitive data hijacking, access to unauthorized information, web page's content modification, command execution, etc. Therefore it is fundamental to protect web applications and to adopt the suitable security methods.

Unfortunately, conventional firewalls, operating at network and transport layers, are usually not enough to protect against web-specific attacks. To be really effective, the detection is to be moved to the application layer.

An Intrusion Detection System (IDS) analyzes information from a computer or a network to detect malicious actions and behaviors that can compromise the security of a computer system. When a malicious behavior is detected, an alarm is launched. Traditionally, IDS's have been classified as either signature detection systems (also called negative approach) or anomaly detection systems (positive approach). An hybrid intrusion detection system combines the techniques of the two approaches.

The signature-based approach looks for the signatures of known attacks (misuse of the system resources), which exploit weaknesses in system and application software. It uses pattern matching techniques against a frequently updated database of attack signatures. It is useful to detect already known attacks or their slight variations, but not the new ones or malicious variations that defeat the pattern recognition engine.

The anomaly-based approach looks for behavior or use of computer resources deviating from "normal" or "common" behavior [1]. The underlying principle of this approach is that "attack behavior" differs enough from "normal user behavior" thus it can be detected by cataloging and identifying the differences involved. First, the "normal" behavior must be well defined, which is not an easy task. Once normal behavior is fully qualified, irregular behavior will be tagged as intrusive.

One of the major shortcomings of signature-based IDSs is their susceptibility to evasion attacks, such as fragmentation, avoiding defaults, low-bandwidth attacks, or simply pattern changing. Additionally, attack pattern matching systems require large signature databases constantly updated. The comparison of incoming traffic against every signature in the database requires a high computational effort, with the consequence of reduced throughput.

In these scenarios where signature-based IDS's fail, anomaly-based systems permit discerning normal traffic from suspicious activity without signature matching. However, in rather complex environments, such as large networks with multiple servers and different operating systems, to get an up-to-date and feasible picture of what "normal" network traffic should look like, proves to be a hard problem. As another disadvantage, the rate of false positives (events erroneously classified as attacks) in anomaly-based systems is usually higher than in signature based ones. Both approaches, anomaly detection and signature matching, are compared in Table 1, which shows the main features of both security models.

Anomaly detection	**Signature matching**
It contains a complete set of valid requests and they are identified accurately, therefore it is possible to detect new attacks, zero-day attacks and variations of attacks	Only the attacks described in the signatures can be detected. The set of invalid requests is incomplete and their accuracy cannot be defined. As a consequence, new attacks and malicious variations cannot be detected
As the definition of the valid requests is complete and accurate, the	The administrative work is high as the signatures have to be updated to

administrative overhead is low	contain the new attacks and the new variations of the existing attacks
Definition of normal traffic is not an easy task in large and complex web applications	Signatures are easy to develop and understand if the behavior to be identified is known
The normal behavior is defined, it is not needed to define a signature for every attack and their variations	A signature has to be defined for every attack and their variations
It works well against self-modifying attacks	Usually, it does not work very accurately against attacks with self-modifying behavior. Such attacks are usually generated by humans and polymorphic worms
It is not easy to know exactly which issue caused the alert	The events generated by a signature-based IDS can inform very precisely about what caused the alert, which makes it easier to research on the causing issue
The resource usage is low	There is a heavy usage of the resources
It is scalable	It is not scalable

Table 1. Comparison between the anomaly-based and the signature-based security models.

The results of signature-based IDSs depend on the actual signature configuration for each web application, hence it is complicated to compare these results with anomaly-based IDS's ones.

Multiple and varied techniques have been used to solve the general intrusion detection problem [3], such as clustering [4], [5] Markov models [6], [7], neural networks [8], [9], fuzzy logic [10],[11], genetic algorithms [12], [18], artificial immune systems [14], etc. Even though there are still some open challenges in intrusion detection.

Web Application Firewalls (WAF) analyzes the HTTP traffic (application layer) in order to detect malicious behaviors that can compromise the security of web applications.

In this paper, a simple and effective anomaly-based WAF is presented. This system relies on an XML file to describe what a normal web application is. Any irregular behavior is flagged as intrusive. The XML file must be tailored for every target application to be protected. Further details about the system will be explained in the next sections of this paper.

Up to now, some of the most important works developed to solve the web attacks detection problem are [6], [15] and [16].

[6] presents an anomaly-based system which uses Markov chains to model the HTTP traffic. The packet payload is parameterised for the evaluation of the incoming traffic: it is segmented into a certain number of contiguous blocks, which are subsequently quantized according to a previously trained scalar codebook. Then, the temporal sequence of the symbols obtained is evaluated by means of a Markov model obtained from the training phase.

The system presented in [15] is also anomaly-based. It analyzes client queries that reference server-side programs. More precisely, the analysis techniques used by the tool take advantage of the particular structure of HTTP queries that contain parameters. The system creates models for a wide-range of different features of these queries, such as attribute length, attribute character distribution, attribute presence or absence, attribute order, access frequency, inter-request time delay, etc. The access patterns of such queries and their parameters are compared with established profiles that are specific to the program or active document being referenced.

[16] introduces an intrusion detection software component based on text-mining techniques. By using text categorisation, it is capable of learning the characteristics of both normal and malicious user behaviour from the log entries generated by the web application server. This IDS is evaluated on a real web-based telemedicine system.

The rest of the paper is organized as follows. In Sect. 2, concepts of web applications and web attacks are exposed, as well as a description of the most important web vulnerabilities. In Sect. 3 a system overview is given, where system architecture, normal behavior modeling and attack detection are explained. Section 4 refers to experiments. This section includes XML file generation, artificial traffic generation, the training phase, the test phase, WAF protection mechanisms, performance measurement and results. Section 5 describes system limitations and suggests future work. Finally, in Sec. 6, the conclusions of this work are captured.

2. Web Applications and Web Attacks

2.1 Web applications

Web applications are usually divided into three logical tiers: presentation, application and storage. Typically, a web server is the first tier (presentation), an engine using some web content technology is the middle tier (application logic) and finally, a database is the third tier (storage). Some examples can be cited: IIS and Apache are popular web servers, Web Logic Server and Apache Tomcat are well known applications servers and finally, Oracle and MySQL are frequently used databases. Separating the presentation tier from the storage one facilitates the design and the maintenance of the web site. The web server sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

Most of the Web contents are dynamic. Dynamic pages, in contrast to static pages, retrieve its content from the database on demand. There are a number of platform-specific technologies to achieve the on-the-fly generation of web content, such as CGI programming in Perl, Python, or C/C++, as well as JSP, ASP.NET, or PHP pages, programmed in a variety of languages such as Java, VB, C#, etc. Dynamic pages require access to the back-end database where the application information is stored, hence attacks against these pages usually aim at the data stored in the database.

2.2 Web Attacks

Web attacks can be classified as static or dynamic, depending on whether they are common to all web applications hosted on the same platform or depend on the specific application [2].

Static web attacks look for security vulnerabilities in the web application platform: web server, application server, database server, firewall, operating system, and third-party components, such as shopping carts, cryptographic modules, payment gateways, etc. These security pitfalls comprise well-known vulnerabilities and erroneous configurations. There are both commercial and free automated tools capable of scanning a server in search of such known vulnerabilities and configuration errors. A common feature of all these vulnerabilities is that they request pages, file extensions, or elements that do not form part of the web application as intended for the end user. Therefore it is very easy to detect suspicious behavior when any resource which does not belong to the application visible by the user is requested.

Dynamic web attacks only request legal pages of the application but they subvert the expected parameters. Manipulation of input arguments can lead to several attacks with different consequences: disclosure of information about the platform, information about other users theft, command execution, etc. In this case, provided with information regarding the type and range of values expected as user input, it is possible to detect any request trying to manipulate normal inputs.

The WAF presented in this paper incorporates techniques which make possible to detect both static and dynamic web attacks.

In the next subsection some of the most important web attacks are exposed.

2.3 Web Vulnerabilities

There is a large amount of web vulnerabilities. In this subsection the most important ones are explained, considering the frequency of exploitation and the impact. The attacks listed are included in some way in the Owasp Top Ten Project [17, 13], which presents the most critical web application security vulnerabilities. Next, a description of the most relevant web vulnerabilities is presented.

- *Obsolete file existence.* Obsolete files can reveal information about the application and show facilities to access to the server data.
- *Default file or example file existence.* Default and example files make easier the access to the server data.
- *Server source file disclosure.* An attacker could access to the server files.
- *HTTP method validity.* Some HTTP methods allow the modification of the application, hence these methods should never be available for an attacker.
- *CRLF injection.* By including control characters used by operating systems to indicate the end of a line, it is possible to execute illegal commands in the system.
- *Failure to restrict URL Access.* Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly.
- *Invalid parameters.* Parameters in the URL or in the body of the request can be manipulated. If the information from web requests is not validated before being used by a web application, attackers can use these flaws to attack backside components through a web application.
- *Command injection.* Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.
- *Cross site scripting (XSS).* It can be perfomed when an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute javascript in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.
- *SQL injection.* It occurs in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed in the database.
- *Buffer overflows.* Web application components in some languages that do not properly validate input (in this case, specially its length) can be forced to crash and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.
- *Broken Authentication and Session Management.* Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities.
- *Broken Access control.* Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.
- *Remote administration flaws.* Many web applications allow administrators to access the site using a web interface. If these administrative functions are not very carefully protected, an attacker can gain full access to all aspects of a site.
- *Web application and server misconfiguration.* Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not always secure after the default installation.
- *Malicious File Execution.* Code vulnerable to remote file inclusion allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise. Malicious file execution attacks affect PHP, XML and any framework which accepts filenames or files from users.
- *Insecure Direct Object Reference.* A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to

access other objects without authorization.

- *Information Leakage and Improper Error Handling.* Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks.

Attacks exploiting these vulnerabilities will be used to test the performance of the WAF being presented, thus these attacks are included in the malicious traffic generated to test the system. Traffic generation is explained in Sec. 4.3 and the test phase is described in Sec. 4.5. The mechanisms used by the WAF to protect all the before mentioned vulnerabilities are listed in Sec. 4.6.

3. System Overview

3.1 Architecture

Our WAF analyzes HTTP requests sent by a client browser trying to get access to a web server. The analysis takes place exclusively at the application layer. The system follows the anomaly-based approach, detecting known and unknown web attacks, in contrast with existing signature-based WAFs. ModSecurity [19] is a popular signature-based WAF.

In our architecture, the system operates as a proxy located between the client and the web server. Likewise, the system might be embedded as a module within the server. However, the first approach enjoys the advantage of being independent of the web platform.

This proxy analyzes all the traffic sent by the client. The input of the detection process consists of a collection of HTTP requests $\{r_1, r_2, \ldots r_n\}$. The output is a single bit a_i for each input request r_i, which indicates whether the request is normal or anomalous. The proxy is able to work in two different modes of operation: as an IDS or as an IPS.

In detection mode, the proxy simply analyzes the incoming packets and tries to find suspicious patterns. If a suspicious request is detected, the proxy launches an alert; otherwise, it remains inactive. In any case, the request will reach the web server. When operating in detection mode, attacks could succeed, whereas false positives do not limit the system functionality.

In prevention mode, the proxy receives requests from clients and analyzes them. If the request is valid, the proxy routes it to the server, and sends back the received response to the client. If not, the proxy blocks the request, and sends back a generic denied access page to the client. Thus, the communication between proxy and server is established only when the request is deemed as valid.

A diagram of WAF's architecture is shown in Fig. 1.

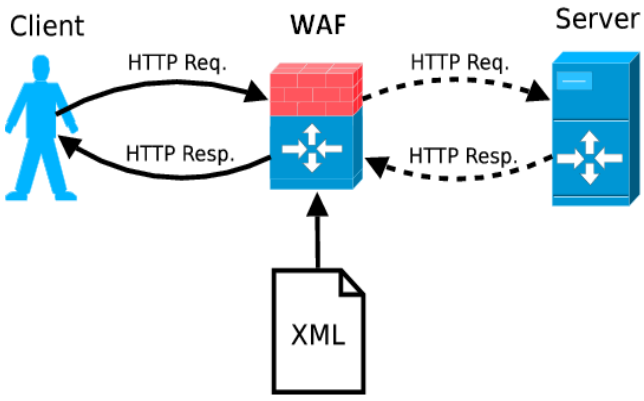

Figure 1. Web Application Firewall architecture

3.2 Normal Behavior Description

Prior to the detection process, the system needs a precise picture of what the normal behavior is in a specific web application. For this purpose, our system relies on an XML file which contains a thorough description of the web application's normal behavior. Once a request is received, the system compares it with the normal behavior model. If the difference exceeds the given thresholds, then the request is flagged as an attack and an alert is launched.

The XML file contains rules regarding to the correctness of HTTP verbs, HTTP headers, accessed resources (files), arguments, and values for the arguments. This file contains three main nodes:

Verbs. The *verbs* node simply specifies the list of allowed HTTP verbs.

Headers. The *headers* node specifies a list of some HTTP headers and their allowed values.

Directories. The *directories* node has a tree-like structure, in close correspondence to the web application's directory structure.

1. Each directory in the web application space is represented in the XML file by a *directory* node, allowing nesting of directories within directories. The attribute *name* defines these nodes.
2. Each file in the web application space is represented by a *file* node within a *directory* node and is defined by its attribute *name*.
3. Input arguments are represented by *argument* nodes within the corresponding *file* node. Each argument is defined by its name and a boolean value *requiredField* indicating whether the request should be rejected if the argument is missing.
4. Legal values for arguments should meet some statistical rules, which are represented by a *stats* node within the corresponding *argument* node. These statistical properties together give a description of the expected values.

Each relevant property is defined by an attribute within the *stats* node. In our approach we considered the following relevant properties:

- *special*: set of special characters (different from letters and digits) allowed
- *lengthMin:* minimum input length
- *lengthMax:* maximum input length

- *letterMin:* minimum percentage of letters
- *letterMax:* maximum percentage of letters
- *digitMin:* minimum percentage of digits
- *digitMax:* maximum percentage of digits
- *specialMin:* minimum percentage of special characters (of those included in the property "special")
- *specialMax*: maximum percentage of special characters (of those included in the property "special")

These properties allow the definition of four intervals (length, letters, digits and special characters) for the allowed values for each argument. Only the values within the interval are accepted. Requests with argument values exceeding their corresponding normal intervals will be rejected.

The adequate construction of the XML file with the suitable intervals is crucial for a good detection process. An example of XML configuration file is shown in Fig. 2.

```
<?xml version="1.0" encoding="iso-8859-1"
standalone="no"?>

<configuration>
  <verbs>
    <verb>GET</verb>
    <verb>POST</verb>
  </verbs>
  <headers>
    <rule name="Accept-Charset"
         value="ISO-8859-1"/>
  </headers>
  <directories>
    <directory name="shop">
      <file name="index.jsp"/>
    <directory name="public">
      <file name="add.jsp">
        <argument name="quantity"
                  requiredField="true">
          <stats maxDigit="100"
                  maxLength="3"
                    maxLetter="0"
                  maxOther="0"
                  minDigit="100"
                  minLength="1"
                  minLetter="0"
                  minOther="0"
                  special="" />
        </argument>
        <argument name="product_name"
                  requiredField="true">
          <stats maxDigit="0"
                  maxLength="15"
                  maxLetter="92.94"
                  maxOther="10.01"
                  minDigit="0"
                  minLength="5"
                  minLetter="89.91"
                  minOther="7.15"
                  special="" />
       <argument name="price"
                  requiredField="true">
          <stats maxDigit="100"
                  maxLength="3"
                  maxLetter="0"
                  maxOther="0"
                  minDigit="100"
                  minLength="1"
                  minLetter="0"
                  minOther="0"
                  special="" />
          ...
  </directories>
</configuration>
```

Figure 2. XML file example

3.3 Detection Process

In the detection process, our system follows an approach of the form "deny everything unless explicitly allowed", also known as *positive* security model.

The detection process takes place in the proxy. It consists of several steps, each constituting a different line of defense, in which the different parts of the request are checked with the aid of the XML file. If an incoming request fails to pass one of these lines of defense, an attack is assumed: a customizable error page is returned to the user and the request is logged for further inspection. It is important to stress that these requests will never reach the web server when operating in prevention mode.

The detection process is composed of the following steps:

1. Verb check. The verb must be present in the XML file, otherwise the request is rejected. For example, in the applications in which only GET, POST and HEAD are required to work correctly, the XML file could be configured accordingly, thus rejecting requests that use any other verb.
2. Headers check. If the header appears in the XML file, its value must be included too. Different values will not be accepted, thus preventing attacks embedded in these elements.
3. Resource test. The system checks whether the requested resource is valid. For this purpose, the XML configuration file contains a complete list of all files that are allowed to be served. If the requested resource is not present in the list, a web attack is assumed.
4. Arguments test. If the request has any argument, the following aspects are checked:

a) It is checked that all arguments are allowed for the resource. If the request includes arguments not listed in the XML file for the corresponding resource, a manipulation of the arguments is assumed and thus the request is rejected.

b) It is confirmed that all mandatory arguments are present in the request. If any mandatory argument (requiredField="true") is not present in the request, it is rejected.

c) Argument values are cheched. An incoming request will be allowed if all parameter values are identified as normal. Argument values are decoded before being checked. For the argument value test, the statistical properties of the corresponding argument are used. If any property of the argument is outside the corresponding interval or contains any forbidden special character, the request is rejected.

These steps allow the detection of both static attacks, which request resources that do not belong to the application, and dynamic attaks, which manipulate the arguments of the request. Figure 3 depicts the detection process.

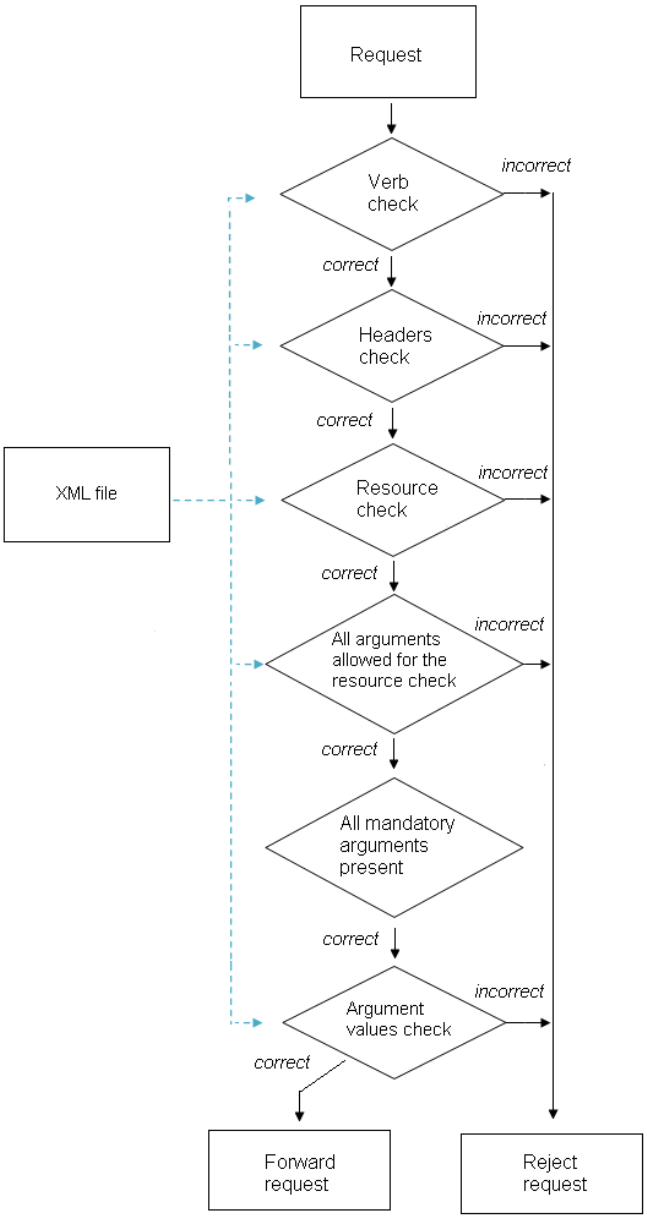

Figure 3. Detection process flow

4. Experiments

4.1 Case Study: Web Shopping

The WAF has been configured to protect a specific web application, consisting of an e-commerce web store, where users can register and buy products using a shopping cart.

4.2 XML File Generation

As already stated, the XML file describes the normal behavior of the web application. Therefore, to train the system and configure this file, only normal (non-malicious) traffic directed to the target web application is required. Nevertheless, how to obtain only normal traffic may not be an easy task. To obtain good detection results thousands of requests are needed. There are some alternatives to obtain normal traffic:

1. Thousands of legitimate users can surf the target web application and generate normal traffic. However, getting thousands of people to surf the web might not be easy.
2. The application can be published in the Internet. Unfortunately, attacks will be mixed with normal traffic, so this traffic cannot be used to train the system.
3. Traffic can be generated artificially. Although the traffic is not real, we can be sure that only normal traffic is included.

For our purposes, we considered artificial traffic generation to be the most suitable approach.

4.3 Artificial Traffic Generation

In our approach, normal and anomalous request databases are generated artificially with the help of dictionaries.

Dictionaries. Dictionaries are data files which contain real data to fill the different arguments used in the target application. Names, surnames, addresses, etc., are examples of dictionaries used.

A set of dictionaries containing only allowed values is used to generate the normal request database. A different set of dictionaries is used to generate the anomalous request database. The latter dictionaries contain both known attacks and illegal values with no malicious intention.

Normal Traffic Generation. Allowed HTTP requests are generated for each page in the web application. Arguments and cookies in the page, if any, are also filled in with values from the normal dictionaries. The result is a normal request database (*NormalDB*). Some requests from *NormalDB* will be used in the training phase and some others will be used in the test phase.

Anomalous Traffic Generation. Illegal HTTP requests are generated with the help of anomalous dictionaries. Examples of the attacks trying to exploit the vulnerabilities listed in Sec. 2.3 are included in the anomalous traffic in order to test the system. There are three types of anomalous requests:

1. **Static attacks** fabricate the resource requested. These requests include attacks like obsolete files, configuration files, default files, etc.
2. **Dynamic attacks** modify valid request arguments: SQL injection, cross-site scripting, invalid parameters, command injection, buffer overflows, broken authentication and session tampering, etc.
3. **Unintentional illegal requests**. These requests should also be rejected even though they do not have malicious intention.

The result is an anomalous request database (*AnomalousDB*), which will be used only in the test phase.

4.4 Training Phase

During the training phase, the system learns the web application normal behavior. The aim is to obtain the XML file from normal requests. In the construction of the XML file, different HTTP aspects must be considered:

- Argument values are characterized by extracting their statistical properties from the requests.
- Verbs, resources and certain headers found in the requests are included directly in the XML file as allowed elements.

4.5 Test Phase

During the test phase, depicted in Fig. 4, the proxy accepts requests from both databases, *NormalDB* and *AnomalousDB*,

and relies on the XML file to decide whether the requests are normal or anomalous. Considering the amount of correctly and incorrectly classified requests, the performance of the system can be measured and the results obtained.

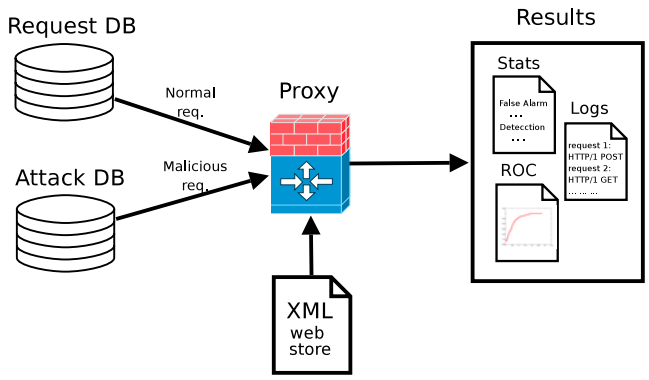

Figure 4. System test phase

4.6 WAF Protection Mechanisms

As previously stated, attacks trying to explote the vulnerabilities described in Sec.2.3 have been included into the malicious traffic (Sec. 4.3) and they have been used to test the performance of the system.

When the normal behavior is correctly identified, all the previously mentioned attacks can be detected (and therefore all the vulnerabilities protected) by the WAF presented in this paper. This section shows how the characteristics and mechanisms used by the WAF are effective to protect against these attacks. The WAF's mechanisms can detect both static and dynamic attacks.

When allowed directories and files are completely specified, the system protects against third party misconfiguration and known vulnerabilities. Attacks against these vulnerabilities are usually very well documented and publicized. They rely on requesting resources present by default in web servers which a legitimate user will never request directly and thus are easy to spot. Therefore, the directories and files enumeration can prevent from attacks exploiting obsolete file existence, default file or example file existence, server source file disclosure, HTTP method validity, failure to restrict URL access, web application and server misconfiguration, etc.

Attacks manipulating parameters are fenced off by the proper definition of the statistical intervals. In the case of buffer overflow, the length property is of paramount importance. Many attacks make use of special characters (typically different from letters and from digits) in order to perform malicious actions. For instance, this is the case of SQL injection, which uses characters with special meaning in SQL to get queries or commands unexpectedly executed. For this reason, the minimum and maximum percentage of letters, digits and special characters are crucial for recognizing these attacks. Even more, any special character present in the input argument is not allowed unless it is included in the property called "special". The interval check help to frustrate attacks exploting vulnerabilities such as CRLF injection, invalid parameters, command injection, XSS, SQL injection, buffer overflow, broken authentication and session management, etc.

4.7 Performance measurement

The performance of the detector is then measured by Receiver Operating Characteristic (ROC) curves [20]. A ROC curve plots the attack detection rate (true positives, *TP*) against the false alarm rate (false positives, *FP*).

$$DetectionRate = \frac{TP}{TP + FN} \quad (1)$$

$$FalseAlarmRate = \frac{FP}{FP + TN} \quad (2)$$

The parameter of the ROC curve is the number of requests used in the training phase.

4.8 Results

Several experiments have been performed using an increased amount of training requests in the training phase. For each experiment, the proxy received 1000 normal requests and 1000 attacks during the test phase.

Figure 5 (a) and (b) show the results obtained by the WAF while protecting the tested web application. As can be seen in Fig. 5 (a), very satisfactory results are obtained: the false alarm rate is close to 0 whereas the detection rate is close to 1. As shown in Fig. 5 (b), at the beginning, with a low amount of training requests, the proxy rejects almost all requests (both normal and attacks). As a consequence, the detection rate is perfect (1) and the false positive rate is high. As the training progresses, the false alarm rate decreases quickly and the detection rate remains reasonably high.

Therefore, this WAF is adequate to protect from web attacks due to its capacity to detect different attacks generating a very little amount of false alarms.

It is important to notice that when the XML file closely characterizes the web application normal behavior, the different kinds of attacks can be detected and few false alarms are raised.

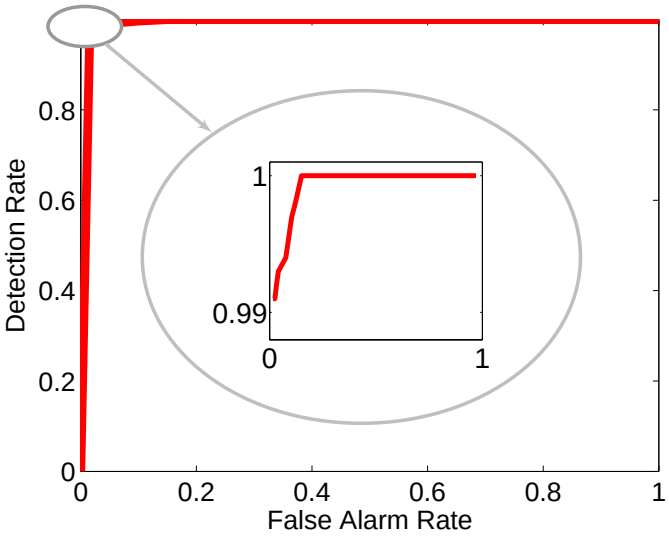

Figure 5 (a). ROC curve of WAF protecting the web store

Figure 5 (b). The false alarm rate and the detection rate vs the number of training requests is plotted

5. Limitations and Future Work

As shown in the previous section, when the XML file is configured correctly, the system succeeds in detecting any kind of web attacks. Thus, the main issue is how to automatically configure the XML description file. In our approach, the XML file is built from a set of allowed requests in the target web application. An important advantage of this solution is that the XML file is built automatically from the normal traffic, thus the directory structure of the web application and the statistical characterization of the arguments is automatically inferred from the input requests. However, obtaining only normal and non-malicious traffic may not be an easy task, as was discussed in Sec. 4.2. Therefore, the main limitation consists in correctly implementing the training phase for any web application.

Other limitations arise when protecting complex web applications. For instance, web sites that create and remove pages dynamically, generate new URLs to access resources, or allow users for updating contents, may difficult the XML file configuration. Further modifications of the system will attempt to solve these problems, by statistically characterizing the URLs of allowed resources.

Future work refers to signing cookies and hidden fields in order to avoid cookie poisoning and hidden field manipulation attacks. Also, URL patterns will be used in describing sites with dynamic resources.

6. Conclusions

We presented a simple and efficient web attack detection system or Web Application Firewall (WAF). As the system is based on the anomaly-based methodology it proved to be able to protect web applications from both known and unknown attacks. The system analyzes input requests and decides whether they are anomalous or not. For the decision, the WAF relies on an XML file which specifies web application normal behavior.

The experiments show that as long as the XML file correctly defines normality for a given target application, near perfect results are obtained. Thus, the main challenge is how to create an accurate XML file in a fully automated manner for any web application. We show that inasmuch great amounts of normal (non-malicious) traffic are available for the target application, this automatic configuration is possible using a statistical characterization of the input traffic.

Acknowledgement

We would like to thank the Ministerio de Industria, Turismo y Comercio, project SEGUR@ (CENIT2007-2010), the Ministerio de Ciencia e Innovacion, project CUCO (MTM2008-02194), and the Spanish National Research Council (CSIC), programme JAE/I3P.

References

[1] García-Teodoro P., Díaz-Verdejo J., Maciá-Fernández G., Vázquez E. Anomaly-based network intrusion detection: Techniques, systems and challenges, Computers and Security, 28, 1-2, 18-28 (2009)

[2] Alvarez G., Petrovic S.: A new taxonomy of Web attacks suitable for efficient encoding, Computers and Security, 22, 5, 453-449 (2003)

[3] Patcha, A., Park J.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks. 51, 12, 3448–3470 (2007)

[4] Petrović S., Álvarez G., Orfila A., Carbó J.: Labelling Clusters in an Intrusion Detection System Using a Combination of Clustering Evaluation Techniques. Proceedings of the 39th Hawaii International Conference on System Sciences, 129b-129b (8 pages), IEEE Computer Society Press, Kauai, Hawaii, USA (2006)

[5] Pouget F., Dacier M., Zimmerman J., Clark A., Mohay G.: Internet Attack Knowledge Discovery via Clusters and Cliques of Attack Traces. Journal of Information Assurance and Security, 1, 21-32 (2006)

[6] Estévez-Tapiador J., García-Teodoro P., Díaz-Verdejo J.: Measuring normality in HTTP traffic for anomaly-based intrusion detection. Computer Networks, 45, 2, 175–193 (2004)

[7] Tokhtabayev A., Skormin V.: Increasing Confidence of IDS through Anomaly Propagation Analysis. Journal of Information Assurance and Security, 2, 107-116 (2007)

[8] Tong X. , Wang Z. , Yu H. A research using hybrid RBF/Elman neural networks for intrusion detection system secure model. Computer Physics Communications 180, 10, 1795–1801 (2009)

[9] Yang Y., Jiang D. ,Xia M.: Using Improved GHSOM for Intrusion Detection. Journal of Information Assurance and Security, 5, 232-239 (2010)

[10] Tajbakhsh A., Rahmati M., Mirzaei A. Intrusion detection using fuzzy association rules. Applied Soft Computing, 9, 2, 462-469(2009)

[11] Akbarpour Sekeh M., Aizaini bin Maarof M., Yazid Idris M., Motiei M.: Fuzzy Intrusion Detection System via Data Mining Technique With Sequences of System Calls. Journal of Information Assurance and Security, 5, 224-231 (2010)

[12] Saniee Abadeh M., Habibi J., Barzegar Z., Sergi M. A parallel genetic local search algorithm for intrusion detection in computer networks. Engineering Applications of Artificial Intelligence, 20, 8, 1058-1069 (2007)

[13] Banković Z., Moya J.M., Araujo A., Bojanić S., Nieto-Taladriz O.: A Genetic Algorithm-based Solution for

Intrusion Detection. Journal of Information Assurance and Security, 4, 192-199 (2009)
[14] Aickelin U., Greensmith J. Sensing danger: Innate immunology for intrusion detection. Information Security Technical Report, 12, 4, 218-227(2007)
[15] Kruegel C., Vigna G., Robertson W.: A multi-model approach to the detection of web-based attacks. Computer Networks, 48, 5, 717–738 (2005)
[16] García Adeva J. J., Pikaza Atxa J. M.: Intrusion detection in web applications using text mining. Engineering Applications of Artificial Intelligence, 20, 4, 555-566 (2007)
[17] Top 10 2007. Owasp Foundation. http://www.owasp.org/index.php/Top_10_2007 (2009)
[18] Owasp Top Ten Project. Owasp Foundation. http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (2009)
[19] ModSecurity. Open Source signature-based Web Application Firewall. http://www.modsecurity.org (2009)
[20] Provost F., Fawcett T., Kohavi R.: The case against accuracy estimation for comparing induction algorithms. Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, San Mateo, CA (1998)

Author Biographies

Carmen Torrano-Gimenez. The author was born in Murcia (Spain), in 1982. She received the Computer Science Engineering degree from the Complutense University of Madrid (Spain) in 2005. Additionally, she received M.Sc. degree in Computer Science and Technology from the Carlos III University of Madrid (Spain) in 2008. At the moment, she is a Ph.D student at the Spanish National Research Council (CSIC), at the Information Processing and Coding department at the Applied Physics Institute. The author's major fields of study are security in web applications and intrusion detection systems.

Alejandro Perez-Villegas. He was born in Granada (Spain) in 1982. He received his B.S. degree in Computer Science from the University of Granada (Spain) in 2006. He received his M.S. degree in Computer Science and Technology from the University Carlos III of Madrid (Spain) in 2009. At present, he is holding a Ph.D grant from the Spanish National Research Council (CSIC), at the Information Processing and Coding Department at the Applied Physics Institute. The author's research interests are security in web applications and intrusion detection systems.

Gonzalo Alvarez. He was born in the Basque Country (Spain) in 1971. He received his M.S. degree in Telecommunications Engineering from the University of the Basque Country (Spain) in 1995. His Ph.D. degree in Computer Science from the Polytechnic University of Madrid (Spain) was received in 2000. He is a tenured scientist at the Spanish National Research Council (CSIC), at the Information Processing and Coding department at the Applied Physics Institute. He is interested in cryptology and Internet security, fields in which he has a large experience.