Detection of Server-side Web Attacks

Igino Corona IGINO.CORONA@DIEE.UNICA.IT
Giorgio Giacinto GIACINTO@DIEE.UNICA.IT
Department of Electrical and Electronic Engineering, University of Cagliari, Italy

Editors: Tom Diethe, Nello Cristianini, John Shawe-Taylor

Abstract

Web servers and server-side applications constitute the key components of modern Internet services. We present a pattern recognition system to the detection of intrusion attempts that target such components. Our system is anomaly-based, i.e., we model the normal (legitimate) traffic and intrusion attempts are identified as anomalous traffic. In order to address the presence of attacks (noise) inside the training set we employ an ad-hoc outlier detection technique. This approach does not require supervision and allows us to accurately detect both known and unknown attacks against web services.

Keywords: Intrusion Detection, Unsupervised learning, Web services

1. Introduction

Today, most of on-line services are implemented as web applications. On-line banking, web search engines, email applications, social networks are just few examples of such *web* services. Web content is generated in real time by a software application running at server-side, i.e. the so-called web application. For example, a request on the following link `http://www.google.com/search?q=`*wapa*`&hl=`*en* generates a HTML page containing search results for *wapa* in *en* language.

According to this scheme, the operations performed by web applications *depend on* their inputs (e.g. the strings *wapa* and *en*). This scheme is indeed flexible and has been proven very successful to support the information exchange on the Internet. On the other hand, unexpected -well-crafted- inputs may be submitted by cyber-criminals to *remotely* gain confidential data or perform unauthorized operations. Since web services are becoming even more complex and ubiquitous, it is very difficult to encounter all possible exceptions to the expected behavior of web servers and web applications. This explains why server-side web security is currently one of the key problems of the Internet [SANS (2009)].

In this paper we present a pattern recognition system for the detection of attacks against web services. We focus on the detection of attacks targeting input validation vulnerabilities, i.e. the most diffused and threatening attacks [SANS (2009)]. As in related work [Kruegel et al. (2005), Bolzoni and Etalle (2008), Guyet et al. (2009), Krueger et al. (2010)], we assume that by collecting a large enough sample of real web requests on a web server, it is possible to describe the legitimate (normal) profile of web server's inputs. Our goal is to distinguish between attacks and legitimate requests, but also to identify different attack categories. Thus, we identify a set of discriminant features and for each request feature a model of its legitimate profile is built. An attack may be identified by an anomalous value in

© 2010 I. Corona & G. Giacinto.

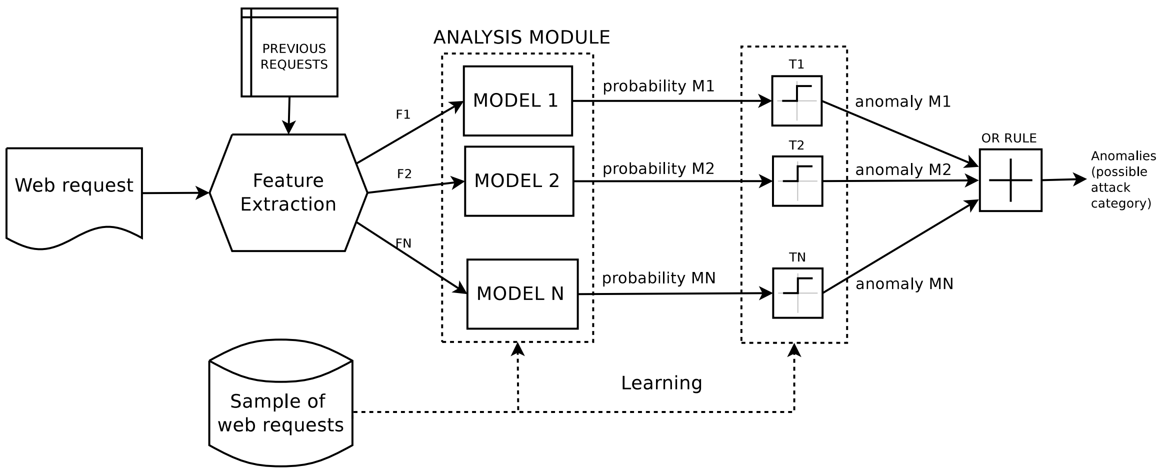

Figure 1: Basic scheme of the proposed Intrusion Detection System.

any of the predefined features. Each model may be considered as an independent anomaly sensor, and the vector of models' outputs may be used to infer the attack class.

The paper is structured as follows. Section 2 outlines the general architecture of our system. Section 3 describes the system's training process. In Section 4 we present more in detail features and models used to describe the legitimate profile of web requests. In Section 5 we present the performance of our system on a production web server. Finally, in Section 6 conclusions are drawn.

2. Architecture

Figure 1 depicts the basic architecture of our Intrusion Detection System (IDS). It is composed by multiple, independent, anomaly sensors. Each sensor is dedicated to the detection of anomalies in a specific *feature* of web requests, since each feature, independently, could be useful to spot an attack pattern. Also, the set of detected anomalies may be employed to infer the attack category (or identify new attack categories). Each sensor is composed by a statistical model of the legitimate (normal) feature's values and a decision module based on a probability threshold. Whenever a test request reaches the web server, the probability (of normality) of each feature's value is assigned by the corresponding model. Then, the corresponding decision module applies a suitable threshold to the output of the model to establish whether the feature's value is anomalous or not. The request is flagged as an attack if it has at least one anomalous feature (i.e. we employ a OR rule).

3. Training

Legitimate users, i.e. users that employ web services as expected, are typically in higher number than malicious users, i.e. cyber-criminals that try to exploit web service's vulnerabilities to their own benefit. Thus, an enough large set of requests towards a web server may be considered as representative of legitimate behavior (training set). Nevertheless, some attacks, i.e. noise, may be present in such a set. Due to the ad-hoc nature of web services, this noise cannot be easily filtered out considering known attacks, or through manual inspection. To address this issue, for each request feature we identify and filter out outlier's values. Then, the remaining feature values are employed to build the corresponding statistical model. Finally, for each model, we set up the probability threshold so that all filtered training samples are classified as legitimate. The outlier detection algorithm, presented in Section 3.1, is independent on the considered features or models. Thus new traffic features and models may be easily employed by our system.

3.1. Outlier detection algorithm

Let us consider a training set $S = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$ composed by N patterns. A statistical model m is built using the entire set S. Let $p[\mathbf{X} = \mathbf{x}|m]$ be the *likelihood* given by the model m to the pattern $\mathbf{x} \in S$. Now, let us define the *relative* distance measure δ between two patterns $\mathbf{x_j}$ and $\mathbf{x_k}$ as

$$\delta(\mathbf{x_j}, \mathbf{x_k}) = |p[\mathbf{X} = \mathbf{x_j}|m] - p[\mathbf{X} = \mathbf{x_k}|m]| \qquad \mathbf{x_j}, \mathbf{x_k} \in S$$

The training set can be expressed as $S = T \cup O$, where T is the set of target patterns and O is the set of outlier patterns (attacks, noise). In order to identify these two sets, we apply a single-linkage clustering [Jain et al. (1999)], where the distance between two patterns is defined by $\delta(\mathbf{x_j}, \mathbf{x_k})$ and the number of clusters is two. This choice is in agreement with the notion of outlier detection, where the distance δ between a target pattern and a outlier pattern is expected to be higher than the distance δ between two target patterns or two outlier patterns. We validate the clusters T and O by verifying that $|T| \gg |O|$ (e.g. $|T| > 10 \cdot |O|$), since the *majority* of patterns inside S are assumed as target patterns. Afterwards, a new model m' can be trained using only patterns in T, which is considered as the *filtered* training set.

4. Request features and models

Let us refer to the following example, that lights out the general format of a HTTP request message:

Request fields	*Request message (example)*
– – – – – – – – – – – – – – – – – →	
web application	`GET /search?q=wapa&hl=en HTTP/1.1`
http version	`Host: www.google.com`
method	`User-Agent: Mozilla/5.0`
header	`Accept: text/html`
header input value	`Accept-Language: en`
web application attribute	`Accept-Charset: utf-8`
attribute input value	`Keep-Alive: 300`
	`Connection: keep-alive`
	`Referer: http://www.google.com/`
	`Cookie: NID=34=Y7dBzu40Q2eiPOOwXKTZ`

Table 1 summarizes the discriminant features employed by our system. For each feature, we describe the principal category of attacks we may detect, as well as the employed model. Each model, enumerated as A, B, C, may be applied to infer the statistical profile of different features. Models are described in Sections 4.1 4.2, 4.3.

Table 1: Summary of the discriminant features employed by our system. For each feature, we specify the category of attacks that may be detected, and the employed model.

Feature	**Attacks**	**Model**
Sequence of web application attributes	any attack that injects malicious input on "uncommon" attributes for the web application	A (a model for each web application)
Sequence of input characters on web application attributes	input validation attacks leveraging on a specific attribute	A (a model for each web app. attribute)
Frequency of requests on web applications	automated queries performed by software robots	B (a model for each web application)
Request Method	information gathering / buffer overflow attacks exploiting this field	C
HTTP version	information gathering / buffer overflow attacks exploiting this field	C
Request headers: input length	buffer overflow attacks leveraging on the header input	B (a model for each request header)
Request headers: non-alphanumeric input characters	code injection attacks leveraging on the header input	C (a model for each request header)
Request headers: allowed input characters (digit or alphabetic or alphanumeric)	information gathering attacks leveraging on the header input	C (a model for each request header)
Ratio between rejected and total requests (per source IP address)	information gathering and automated discovery of common (vulnerable) web applications	B

4.1. Model A: sequence of symbols

A sequence of symbols is described through a Hidden Markov Model. The training phase is based on the well-known Baum-Welch algorithm [Rabiner (1989)]. We compute the number of states as the average number of distinct symbols of each training sequence. Moreover, we randomly initialize the state transition and the symbol emission matrices. Finally, we build the dictionary of symbols by extracting them from training sequences. Indeed, these choices are in some way arbitrary. However, they allowed us to attain excellent results in previous work [Corona et al. (2009)] and do not require *a-priori* knowledge about the structure of training sequences.

We train the HMM on the (raw) training set S. Subsequently, in order to identify outlier samples inside the training set (i.e. noise), we apply the algorithm described in Section 3.1. Finally, we (re)train the HMM using the filtered training set.

Given an observed sequence $\mathbf{x}$, we obtain the most probable state sequence by using the Viterbi Algorithm [Rabiner (1989)]. Then, the probability of the sequence

$$p[\text{x is legitimate}|\text{model-a}]$$

is computed by combining state transition and the symbol emission matrices, according to the most probable state sequence and the sequence of symbols under analysis.

4.2. Model B: statistical distribution of a non-negative integer value

This model is built from a training set $S = \{x_1, x_2, \ldots, x_N\}$, where each element is a non-negative integer value, i.e., $x_i \in \mathbb{N}_0$. In practice, according to the selected features, the higher x_i, the lower the likelihood that it is legitimate. So, we define the probability of a certain value x as:

$$\begin{cases} p[\text{x is legitimate}|\text{model-b}] = p[|X-\mu| \geq |x-\mu|] = \frac{\sigma^2}{(x-\mu)^2} & if \quad x \geq \mu + \sigma \\ p[\text{x is legitimate}|\text{model-b}] = 1 & \text{otherwise} \end{cases} \tag{1}$$

Since large values should receive lower likelihood, we compute the probability that the random variable X, having mean μ and variance σ^2 exceeds the x value. To this end, we use the well-known Chebyshev inequality $p[|X-\mu| \geq q] \leq \frac{\sigma^2}{q^2}$, with $q = |x-\mu|$ and consider the upper limit of $p[|X-\mu| \geq |x-\mu|]$.

Initially, the model is based on the mean μ and the variance σ^2 of values inside the (raw) training set S. Then we apply the outlier detection algorithm described in Section 3.1. Finally a new model is built considering the filtered training set.

4.3. Model C: statistical distribution of symbols

This model is built from a training set $S = \{x_1, x_2, \ldots, x_N\}$ where each element is a symbol. For a generic symbol x we assign the following probability:

$$p[\text{x is legitimate}|\text{model-c}] = \frac{count(x)}{N} \tag{2}$$

that is, the relative frequency $count(x)$ of x in the training set S. Then we apply the outlier detection algorithm described in Section 3.1. The filtered set of symbols T is used to build a new model following the previous eq. 2. Any symbol not in T is considered as anomalous.

5. Experiments

We experimented with about 450,000 requests, collected from a production web server of our academic institution, in a week time-interval. We subdivided this dataset (Λ) in two datasets: Σ composed by the first 200,000 requests of Λ and T containing all remaining requests. Dataset Σ has been used to train our detection system[1].

We labeled all requests inside Λ with the help of models built by our system. Recall that attack samples are expected to receive lower probability with respect legitimate samples (see Section 3). So, for each feature, we manually inspected samples receiving the lowest

1. In is worth noting that we may not randomly split dataset Λ. This because the last feature in Table 1 depends on the sequence of requests during the time. The training phase required 2 hours and 53 minutes on a common laptop with Linux OS.

likelihood. On the basis of our experience and by employing contextual information[2], we labeled such samples as legitimate or attacks. Such an approach (used also in our previous work [Corona et al. (2009)]) allowed us to find 232 attack requests in Λ. We labeled as legitimate all other web requests in Λ. It is worth noting that the employed approach is effective to label input validation attacks (the focus of this work), while some other attacks may go unnoticed.

The IDS detected autonomously **all** attacks inside the set Λ. A portion of them (102 attacks), was among the requests employed to train the IDS itself (Σ). On the other hand, our system raised a total of 1,252 false alarms (i.e. a false alarm rate of 0.28%) on set Λ.

To better evaluate the detection rate of our system, we built a set of input validation attacks Φ against both web server and web applications of the portal. We then evaluated how many of them were actually detected by the IDS. These attacks have been thoroughly suited to the specific configuration of web server and web application inputs [Corona (2010)]. Dataset Φ accounts for 507 attacks against the web portal under analysis. We analyzed these well-crafted attack requests with our system, and 505 (99.6%) of them have been successfully detected. It is worth noting that the two missed attacks were information gathering attacks with low impact on the security of web services under test. Table 2 outlines the performances of our detection system, in terms of (a) detection, (b) false alarm rate and (c) average response time per request, evaluated on datasets Λ and Φ.

These results clearly highlight that our approach can accurately spot attacks even if they are included in the traffic collected to train the system. We detected all attacks encountered in the wild (i.e. within dataset Λ) and almost all our "custom" attacks in the dataset Φ. In addition, a very low false positive rate is generated (0.2-0.3%). Nevertheless, most of alerts are actually false alarms, since the attack class is heavily undersampled.

Parameter	**Dataset**	**Value**
detection rate	$\Lambda = \Sigma \cup T$	232 attacks out of 232, 100%, ~39alerts/day
	Φ	505 attacks out of 507, 99.6%
false alarm rate	Λ	1,252alerts/447,178reqs, 0.28%, ~209alerts/day
	Σ	450alerts/200,000reqs, 0.22%, ~150alerts/day
	T	802alerts/247,178reqs, 0.32%, ~267alerts/day
response time	Λ	1.2 milliseconds

Table 2: Overview of the performaces of the IDS evaluated on datasets Λ and Φ. Our system has been trained on Σ (the first 200,000 requests of Λ).

6. Conclusions

We presented a pattern recognition system for the detection of attacks against web services. The proposed system is able to learn without supervision from real web traffic towards a web server (training set). An important aspect is that our system handles the presence of attacks in the training set. The experimental evaluation on a production web server

2. For example, we verified legitimate web application inputs by links contained inside web pages, and by browsing as a normal web user.

showed that our IDS is able to attain very good performances, with both high detection and very low false alarm rates. Nevertheless some questions still remain open. IDSs operate in an adversarial environment. We have to evaluate the robustness of our system in front of targeted noise that tries to evade/attack the outlier detection algorithm. Moreover, the false alarm rate may be further reduced, e.g. through a more sophisticate outlier detection algorithm. These aspects will be object of future work.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments on this paper. This material is based upon work supported by the Regione Autonoma della Sardegna, under grant PO Sardegna FSE 2007-2013, L.R.7/2007 "Promozione della ricerca scientifica e dell'innovazione tecnologica in Sardegna".

References

Damiano Bolzoni and Sandro Etalle. Boosting web intrusion detection systems by inferring positive signatures. In *OTM Conferences (2)*, pages 938–955, 2008.

Igino Corona. *Detection of Web-based attacks*. PhD thesis, Cagliari (Italy), 2010. URL `http://prag.diee.unica.it/pra/node/820`.

Igino Corona, Davide Ariu, and Giorgio Giacinto. Hmm-web: a framework for the detection of attacks against web applications. Dresden, Germany, 14/06/2009 2009. URL `http://prag.diee.unica.it/pra/system/files/Corona_ICC2009.pdf`.

Thomas Guyet, Rene Quiniou, Wei Wang, and Marie-Odile Cordier. Self-adaptive web intrusion detection system. *CoRR*, abs/0907.3819, 2009.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/331499.331504.

C. Kruegel, G. Vigna, and W. Robertson. A Multi-model Approach to the Detection of Web-based Attacks. *Computer Networks*, 48(5):717–738, August 2005.

Tammo Krueger, Christian Gehl, Konrad Rieck, and Pavel Laskov. Tokdoc: a self-healing web application firewall. In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1846–1853, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. doi: http://doi.acm.org/10.1145/1774088.1774480.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

SANS. The top cyber security risks - september 2009. Technical report, 2009. URL `http://www.sans.org/top-cyber-security-risks/`.