SuStorID: A Multiple Classifier System for the Protection of Web Services

Igino Corona, Roberto Tronci, Giorgio Giacinto
Dept. of Electrical and Electronic Engineering, University of Cagliari, Italy
{igino.corona,roberto.tronci,giacinto}@diee.unica.it

Abstract

The security of web services is nowadays one of the major concerns for Internet users. Web services may manage confidential information, monetary transactions, or even health-critical systems, such as those employed in public airports or hospitals. A key problem of web services is that they should work as expected even in the presence of malicious inputs. Unfortunately, with the increasing complexity of web services, this task becomes more and more challenging.

In this paper we present SuStorID, a multiple classifier system which is able to model legitimate inputs towards web services, given a sample of web traffic. If anomalous inputs are detected, web services are protected according to a set of anomaly templates. Our experiments, performed on a production environment, highlight that our system can accurately detect web attacks and help security operators to protect their web services against known and unknown attacks.

1. Introduction

The widespread deployment of web services makes them a favorite target for cyber-criminals. The complexity and lack of security training of web developers may expose web services to a wide range of remote attacks. In particular, Internet miscreants may submit malicious, unexpected inputs towards web services to gain their control, obtain confidential information, or infect web visitors in a matter of seconds [6].

Due to the ad-hoc nature of web services, i.e., different services may have different inputs and behavior, defining good rules on web application firewalls is an error-prone activity which often requires a non-negligible human effort. The key point is that "safe" firewall rules should be customized according to expected inputs and behavior of a certain web service.

In this paper we present SuStorID, an intrusion detection system that can be employed to complement web application firewalls to the protection of web services. Given a sample of web traffic towards a certain web service, our system is able to automatically learn the profile of legitimate (normal) inputs. With respect to related work on web security [1], SuStorID is able to address the presence of noise (attacks) within the training set. Moreover, it introduces the concept of anomaly templates, i.e., user-defined models that associate a suitable action in response to anomalous web traffic. Our experiments on a production environment show that SuStorID is able to accurately detect web attacks. We also show that detection accuracy can be significantly improved in few steps through user interaction. SuStorID is released under open-source license and can be freely downloaded at http://comsec.diee.unica.it/sustorid.

2. System Overview

SuStorID presents a multi-classifier architecture, where each classifier is tailored to the detection of a specific class of anomalies within a web request (see Figure 1). SuStorID is structured in a way such that additional classifiers can be easily added into the system to detect new attack classes. This aspect is very important to keep the tool up-to-date, due to the rapid evolution of computer attacks. Moreover, this architecture can help security operators to obtain valuable information about targeted vulnerabilities and thus assess the real treats posed by each anomalous event.

Each classifier monitors a particular field of a request message, e.g., method, web application, HTTP version, (see Figure 2), and establishes whether such a field exhibits anomalies or not. Since each classifier is tailored to detect a specific class of attacks against web services, an alert is raised if at least one classifier detects an anomaly (OR rule).

In order to protect the web services, SuStorID employs a set of predefined models called *anomaly templates*. An anomaly template associates an action (e.g. *log*, *deny request*, or *ignore*) to an anomalous event.

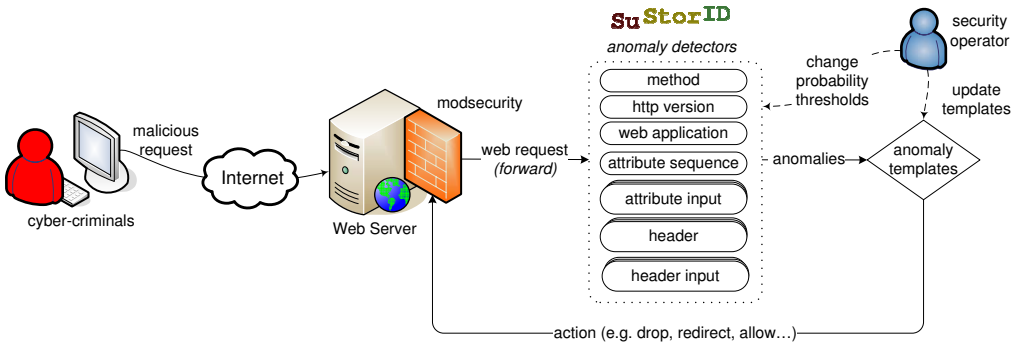

Figure 1. Architecture of SuStorID.

```
GET /search?q=ICPR&hl=en HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Charset: utf-8
Accept-Language: en
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com
Cookie: NID=34=Y7dBzu40Q2eiPOOwXKTZ
```

Figure 2. Example of a HTTP request message. *Message fields:* `web application`, `HTTP version`, `method`, `header`, `header input value`, `web application attribute`, `attribute input value`.

Such rules can be written by a security expert to codify context-sensitive information that is difficult or even impossible to infer from training data. On the other hand, since these rules operate at classification level and classifier outputs have a well-defined meaning, anomaly templates may be easier to write and understand.

3. Learning

One of the biggest problems of web security is to detect unknown malicious requests in a real environment. Typically, the number of normal (legitimate) web requests largely outnumber the number of malicious requests. This is because, usually, most of users employ web services as expected. Thus, SuStorID employs a sample of *real* traffic towards web services to infer the profile of their legitimate inputs. In order to deal with attacks within the training set, an outlier filtering algorithm is applied to *each model*, independently from each other (see Section 3.1). Such an algorithm is "general purpose" in the sense that it can be applied to any classifier, regardless the employed feature set. Thus, no new outlier filtering algorithms are required, if new features/models are added to detect new attacks.

3.1 Outlier filtering 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 $p[\mathbf{x}|m]$ be the *likelihood* given by model m to pattern $\mathbf{x} \in S$. Now, let us define the *relative* distance measure δ between two patterns $\mathbf{x_j}, \mathbf{x_k} \in S$ as

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

The training set can be expressed as $S = T \cup O$, where sets T and O contain target and outlier patterns, respectively. In order to identify these two sets, we apply a single-linkage clustering, where the distance between two patterns is defined by $\delta(\mathbf{x_j}, \mathbf{x_k})$ and the number of clusters is two. We validate the clusters T and O by verifying that $a_t|T| > |O|$, where a_t is the maximum fraction of expected outliers within S. If $a_t|T| > |O|$ is verified, a new model m' is trained using only patterns in T, which is considered as the *filtered* training set. Otherwise, S is considered as composed by target patterns only, i.e., no filtering is performed. The probability threshold for m' is set so that all filtered training samples are classified as legitimate.

3.2 Request features and models

The vast majority of attacks against web applications are realized by submitting inputs having an anomalous structure, length, or unexpected symbols. To highlight such anomalies, SuStorID employs three main models: α, β, γ; they are described in Sections 3.2.1, 3.2.2, 3.2.3, respectively. Each main model can be used to infer the statistical profile of a different traffic feature, and each feature may be useful to spot a different attack technique (see Table 1). For instance, modeling the statistical distribution of legitimate input size allows to detect buffer overflow attacks, which typically show a relatively high size of input strings, while Hidden Markov

Table 1. Summary of the features employed by SuStorID. For each feature, we specify the main category of attacks that may be detected, and its main model.

Feature	Attacks	Model
Sequence of web application attributes	malicious input on "uncommon" attributes	α for each web application
Sequence of input chars on web app. attributes	any input validation attack against a web app. attribute	α for each attribute
Request Method	information gathering / buffer overflow attacks exploiting this field	γ
HTTP version	information gathering / buffer overflow attacks exploiting this field	γ
Request headers: input length	buffer overflow attacks leveraging on header input	β for each request header
Request headers: allowed input characters (digit *or* alphabetic *or* alphanumeric)	code injection attacks leveraging header input	γ for each request header

Models can detect more complex attacks (e.g. such as those against web application attributes), by highlighting an anomalous input structure. For more details about web attack techniques, the interested reader can refer to the Open Web Application Security Project[4].

3.2.1 Model α: sequence of symbols

A sequence of symbols is described through a Hidden Markov Model. The training phase is based on the Baum-Welch algorithm [5]. 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. These choices allowed us to attain very good results in previous work [2] and do not require *a-priori* knowledge about the structure of training sequences. Given an observed sequence $\mathbf{x}$, we obtain the most probable state sequence by using the Viterbi Algorithm [5]. Afterwards, the likelihood of the sequence $p[\mathrm{x}|\alpha]$ is computed by combining state transition and the symbol emission matrices, according to the most probable state sequence for the sequence of symbols under evaluation.

3.2.2 Model β: statistical distribution of a non-negative integer value

The training set $S = \{x_1, \ldots, x_N\}$ is composed by non-negative integer values. We define the probability of a certain value x as:

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

In practice, 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 Chebyshev inequality $p[|X - \mu| \geq q] \leq \frac{\sigma^2}{q^2}$, with $q = |x - \mu|$ and consider the upper limit of $p[|X - \mu| \geq |x - \mu|]$.

Source	Attack techniques	Target
simulation (total: 303)	Cross-site scripting (77), SQL (73), XPath (18), LDAP (18) and Alphabetic (44) Injection, Integer Overflow (40), Server-side Includes (33)	Web server, Web applications, Request Headers
in-the-wild (total: 77)	Brute-force: check for web applications with known bugs (41), SQL Injection (4), Cross-site scripting (19), other Input Validation attacks (13)	Web server, Web Applications, Request Headers

Table 2. *Simulated* attacks and attacks *in-the-wild* against the web services under evaluation.

3.2.3 Model γ: statistical distribution of symbols

The training set $S = \{x_1, \ldots, x_N\}$ is composed by symbols. The probability of a generic symbol x is computed as its relative frequency within S:

$$p[x|\gamma] = |\{x_i : x_i = x, \forall i\}|/N \tag{2}$$

4. Experiments

We collected 51,848 requests (between December 23, 2011 and March 28, 2012) from a web server hosting different public services of the University of Cagliari. Such requests can be roughly subdivided into three main classes: *legitimate*, *attacks in-the-wild* and *simulated attacks*. Legitimate requests are performed by normal users while accessing to web services under evaluation. On the other hand, *attacks in-the-wild* are requests generated by malicious users or (ro)bots, to either find or exploit security vulnerabilities within these web services. Finally, *simulated attacks* are requests that we issued using `WebScarab`[1], in order to thoroughly simulate the behavior of a malicious, skilled user. Table 2 briefly categorize and quantify these attacks. Real attacks were identified by: (a) looking

[1] `WebScarab` is a powerful tool for web vulnerability testing. See `https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project`.

web requests for known attack signatures (see [3]), and (b) training a separate instance of SuStorID and manually checking patterns having lower probability for each model. It is worth notice that step (b) is aimed at detecting unknown/custom attacks against the web services under test, that would otherwise remain undetected, e.g., because no signatures are available for these attacks. This semi-automated procedure is based on SuStorID itself because, to the best of our knowledge, no other equivalent systems are currently available.

4.1 Detection Accuracy and User Interaction

Our evaluation was aimed to point out detection accuracy of SuStorID on a real operating setting. To this end, we trained SuStorID using the *first* 30,000 web requests and evaluated its classification accuracy on the remaining (last) 21,848 requests, according to their timestamps. We set $a_t = 1.5\%$, i.e. no more than 1.5% of attacks are expected within the training patterns given to each model. The whole training phase required about 10.5 h on a computer with Intel Core 2 T7200 processor, 2 GB of RAM and Linux OS kernel 2.6.32. Analyzing a single request (detection of anomalies and anomaly template matching) took 0.358 s on average.

Table 3 highlights the detection accuracy of SuStorID in terms of attack detection rate for simulated (DR^{sim}) attacks, attacks found *in-the-wild* (DR^{wild}), and false alarm rate (FR) without user interaction (UI None) and with an increasing number of anomaly templates (AT) provided by a human operator. We simulated a human operator who setups AT to suppress obvious false alarms, such as requests related to obsolete links, legitimate static resources which never appeared during the training phase, or requests having ad-hoc (rare) headers which do not represent any threat. As it can be seen from Table 3, SuStorID was able to spot *every* attack request within the test set. Nevertheless, without the employment of anomaly templates, SuStorID shows a relatively high FR (5.4%), which may be prohibitive, especially with large amounts of web traffic. On the other hand, most of false alarms can be easily eliminated with the setup of anomaly templates. In our experiments, we were able to narrow down FR from 5.4% to 0.47% by defining 14 anomaly templates (the whole process took about 20 min). This is because many false alarms are easy to be interpreted by a human operator, and they are often duplicate.

UI	FR	$\mathbf{DR^{sim}}$	$\mathbf{DR^{wild}}$
None	1,179/21,848≈5.4%	303/303=100%	77/77=100%
AT: 1	1,088/21,848≈4.98%	303/303=100%	77/77=100%
AT: 2	690/21,848≈3.16%	303/303=100%	77/77=100%
AT: 3	458/21,848≈2.1%	303/303=100%	77/77=100%
AT: 4	448/21,848≈2.05%	303/303=100%	77/77=100%
AT: 5	438/21,848≈2%	303/303=100%	77/77=100%
AT: 6	414/21,848≈1.89%	303/303=100%	77/77=100%
AT: 7	409/21,848≈1.87%	303/303=100%	77/77=100%
AT: 8	401/21,848≈1.83%	303/303=100%	77/77=100%
AT: 9	381/21,848≈1.74%	303/303=100%	77/77=100%
AT: 10	317/21,848≈1.45%	303/303=100%	77/77=100%
AT: 11	155/21,848≈0.71%	303/303=100%	77/77=100%
AT: 12	125/21,848≈0.57%	303/303=100%	77/77=100%
AT: 13	113/21,848≈0.52%	303/303=100%	77/77=100%
AT: 14	103/21,848≈0.47%	303/303=100%	77/77=100%

Table 3. Detection accuracy of SuStorID, evaluated on real web traffic and simulated attacks, for increasing levels of User Interaction (UI).

5. Conclusions

The security of web services is nowadays one of the major concerns for Internet users. In this paper we presented SuStorID, a multiple classifier system to the protection web services. Our experimental results on a production environment highlight that SuStorID can accurately detect web attacks. We also introduce the concept of *anomaly templates*. Such templates can be used to respond to anomalous events, due to known or unknown attacks, in real-time. Our experiments show that they can also be exploited by a human operator to significantly reduce false alarms with little effort.

Acknoledgements This research has been partially carried out within the project "Advanced and secure sharing of multimedia data over social networks in the future Internet" funded by the Regional Administration of Sardinia, Italy (CUP F71J11000690002)

References

[1] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[2] I. Corona, D. Ariu, and G. Giacinto. Hmm-web: a framework for the detection of attacks against web applications. In *IEEE Int. Conf. on Communications (ICC'09)*, pages 747–752, Piscataway, USA, 2009. IEEE Press.

[3] MITRE. Common vulnerabilities and exposures. http://cve.mitre.org, March 2012.

[4] OWASP. http://www.owasp.org.

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

[6] T. Scholte, D. Balzarotti, and E. Kirda. Have things changed now? An empirical study on input validation vulnerabilities in web applications. *"Computers and Security", 2012, ISSN: 0167-4048*, 12 2011.