Anomaly-based Intrusion Detection in Software as a Service

Gustavo Nascimento*, Miguel Correia†
*Portugal Telecom – Portugal
†Instituto Superior Técnico / INESC-ID – Portugal
gustavo-nascimento@telecom.pt, miguel.p.correia@ist.utl.pt

***Abstract*—Anomaly-based intrusion detection systems (IDS) have the ability of detecting previously unknown attacks, which is important since new vulnerabilities and attacks are constantly appearing. Software as a service web applications are currently much targeted by attacks, so they are an obvious application for such IDSs. The paper presents a study of the use of anomaly-based IDSs with data from a production environment hosting a web application of large dimensions. It describes how challenges like processing a large number of requests and obtaining training data without attacks were solved. It also presents an evaluation comparing the accuracy obtained with the different types of models that were used to represent normal behavior.**

I. Introduction

Intrusion detection is the process of identifying malicious activity targeted at computing and networking resources [1]. An intrusion detection system (IDS) monitors computers and/or networks to identify suspicious activity. When such an event is detected, the IDS typically raises an alert. IDSs have been classified as signature-based and anomaly-based. A signature-based (or misuse-based) IDS has a database of attack signatures and works similarly to anti-virus software, by raising an alert when it matches one of the signatures. Those signatures typically address widely used systems or applications for which security vulnerabilities are known. Nevertheless, similarly to anti-virus software that fails to identify viruses when there is no signature available or the virus database is out of date, a signature-based IDS also fails to detect unknown attacks.

To overcome this limitation of signature-based IDSs, researchers have sought other ways to detect intrusions. An *anomaly-based IDS* works by first building a statistical model of usage patterns describing the normal behavior of the monitored resource. After this initial *training phase*, the system uses a similarity metric to compare new input requests with the model, and generates alerts for those deviating significantly, considering them anomalous. Basically, attacks are detected because they produce a different, i.e., anomalous, behavior than what was observed when creating the model. The main advantage of an anomaly-based system is its ability to detect previously unknown (or variants of known) attacks when they appear. Nevertheless, these systems typically suffer from high rates of false positives and can be evaded using mimicry attacks, i.e., attempts to pass as normal behavior, for example by using byte substitution and/or padding techniques.

The popularity of *web applications*, now often designated as *software as a service* (SaaS) when offered by a provider to a set of users, has caught the attention of attackers which try to exploit their vulnerabilities. For example, in a recent survey, 95% of the respondent organizations reported having experienced more than 10 incidents related to their web sites [5]. Therefore, using intrusion detection and specifically anomaly-based intrusion detection is important in such systems. Several anomaly-based IDSs for web applications have already been proposed in the literature [2], [9], [10], [16], [13], [15], [8], [6].

The main goal of our research was to study the use of anomaly-based IDS with data of a production environment hosting a SaaS application of large dimensions, with more than 500,000 requests a day. This poses interesting challenges, like processing of a large number of requests and obtaining training data without attacks. In the literature, anomaly-based IDSs have been tested almost only with simple web applications.

The main contribution of the paper are insights on how to deal with these challenges. The paper first explains how a large collection of requests (14 weeks) was obtained and how part of it was sanitized in order to serve as training data. Then it presents the IDS and the types of models that it supports. The important problem of generalization, i.e., of obtaining models that are more general than the data used to create them, is discussed. Finally, the paper summarizes the evaluation we did of the IDS, mostly comparing the models in order to reach conclusions about which one provides better detection.

II. Related Work

Several anomaly-based detection techniques for web servers and web applications have been proposed in the last decade. All these techniques were based on HTTP, in the sense that the modeling of normal activity and the detection are done by inspecting HTTP messages.

Anomaly-based detectors for web applications were first proposed in [2], that describes a system which uses Bayesian parameter estimation to analyze web access logs and detect anomalous sessions. Their technique assumed that malicious activity expressed itself in the parameters of HTTP requests,

essentially in the URL, which is also what is assumed by most work that followed in the area.

Kruegel et al. proposed a number of similar techniques [9], [10]. These techniques basically consisted of a combination of different detection models: attribute length, attribute character distribution, structural inference, token finding, attribute presence or absence, and attribute order. In this paper we use the models of [9] along with some others.

Wang and Stolfo [16] proposed a system that uses the Mahalanobis distance to detect anomalous requests in data sets with multiple attributes, scaling each variable based on its standard deviation and covariance and taking into account how the measured attributes change in relation to each other.

The approach by Robertson et al. [13] uses heuristics to infer the class of web-based attack. It attempts to address the limitations of anomaly-based intrusion detection systems by using both generalization and characterization techniques. By using generalization, a more abstract description of an anomaly can be created which enables one to group similar attacks. As for characterization, it is used to infer the class of attack that is associated with a group of anomalies.

Wang et al. [15], propose a content anomaly detector based in n-gram analysis, which uses bloom filters and offers resistance to mimicry and polymorphic attacks.

Ingham and Inoue presented a thorough study and comparison of existent anomaly detection methods for HTTP intrusion detection [7], [6]. They suggested two new grammar-based anomaly detection models, DFA and n-grams, described a framework for testing anomaly detection algorithms and presented their findings using data from four different web sites.

This paper uses several of the anomaly-based intrusion detection techniques previously proposed in [7], [6], [9], [16]. However, it aims to present an IDS for a real reasonably complex web application, so not only we implemented several of their techniques, but studied the problem of obtaining good training data and had to deal with several practical issues not considered in these previous works.

III. Obtaining the Traffic Data Sets

The SaaS considered is a popular web application of one of the main Portuguese ISPs, with an average of more than 500,000 requests per day. This application allows users to insert and consult information and pictures about products they want to sell. Most requests are HTTP GETs. Our main restriction was to consider only HTTP GET requests, thus disregarding other requests (mostly POST) and all replies. Replies are not particularly useful for this kind of detection, while requests with POST and other methods accounted just for a very small percentage of the traffic.

A. Initial data extraction

Our data sets were built from a series of network captures of HTTP requests made to this application during a period of 14 weeks. To collect these requests, the web server traffic was port mirrored to a server where it was captured with *TShark*. This tool, a terminal-based version of *Wireshark* [1], is a network packet analyzer that enables capturing packet data from a live network, or reading packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. Its native capture file format is the one used by *libpcap/tcpdump* [2].

When we began this research, we did not use any particular filters with *TShark*. However, and due to the high volume of traffic, associated with the disk space limitations on our capture server, we soon realized that we needed to restrict what was being captured, as the early traces averaged about 8 GB with *bzip2* best compression (-9) per day, which meant around twice the size uncompressed, i.e., 16 GB of daily traffic.

We began by transforming the trace logs into the Common Log Format [11], the format typically used by web servers and used by several of the IDSs mentioned in Section II. For each request, the log entry contained the source IP, the method (GET), the resource requests, a timestamp and a few other data items less relevant for an IDS. Given that the captures obtained with TShark contained the complete requests, and since we deemed the header of the HTTP GET requests to be useful for intrusion detection, we decided to discard those logs and instead produce our data sets differently. These sets were created by using *snort* [14]. This program was not used as an IDS, but instead to reassemble the TCP streams and extract the application-level, i.e., the complete HTTP requests. This allowed our IDS to use the HTTP header lines to test for attacks not contained in the requested resource path. We decided to aggregate the output streams processed by *snort* into weekly data sets.

B. Unfiltered and filtered data sets

We produced two different data sets. The first one contained all the requests resulting from the captures done with *TShark* after being processed by *snort* and parsed by custom scripts to remove incomplete requests. We did not verify this data set for attacks, so we refer it as the *unfiltered data set*.

The second data set – the *filtered data set* – contained much less data. First, it was obtained by selecting the five most accessed sub-applications of the original application and by applying filters to match only the requests directed at those applications. Second, it was filtered to remove attacks, so that it could serve as training data. We selected $n = 4$ different weeks of traffic, each one from a different month, and generated four data slots $Slot[0..n-1]$ to which we applied the algorithm of Figure 1 in order to obtain sanitized data. The slot sets were passed to the sanitization algorithm in order to remove attacks, until the number of detected

[1] www.wireshark.org
[2] www.tcpdump.org

INITIALIZATION

$\forall\, i \in \{0, .., n-1\}\,, NonAttacks[i] = Attacks[i] = \{\}$

SANITIZE-DATASET$(n, Slot)$

```
1  i ← 0
2  repeat
3      TRAIN using Slot[i mod n]\Attacks[i mod n]
4      DETECT over Slot[(i + 1) mod n]\NonAttacks[(i + 1) mod n]
5      for all requests r detected as anomalous do
6          if manual analysis shows that r is an attack then
7              Attack[(i + 1) mod n] ← r
8          else
9              NonAttacks[(i + 1) mod n] ← r
10     i ← i + 1
11 until #attacks detected in last n rounds were less than threshold
12 return ∪Slot[0..n − 1]\ ∪ Attacks[0..n − 1]
```

Figure 1. Algorithm used to obtain a data set without attacks

Data set	Unfiltered size	Filtered size
2010-07-12∗	729058	42659
2010-07-19	751170	38682
2010-07-26	805296	37017
2010-08-02	549256	41598
2010-08-09	630795	40020
2010-08-16∗	555708	48533
2010-08-23	583201	47086
2010-08-30	538670	42552
2010-09-06	615254	43922
2010-09-12	572264	53634
2010-09-20∗	661150	53766
2010-09-27	681352	38851
2010-10-04	608760	43145
2010-10-11∗	657060	39607

Table I
THE WEB SERVER DATA SET SIZES (IN NUMBER OF REQUESTS).

anomalies dropped below a certain threshold. Defining the threshold depends on the environment and data in question, but this algorithm can also easily be adapted to perform the sanitization using a fixed number of iterations (simply by changing the *until* condition).

Table I shows the number of HTTP GET requests in each of the weekly unfiltered and filtered data sets. The four rows marked with a $*$ represent the portion of data that was used to train the anomaly detection models. The remaining rows of the filtered column, represent the traffic that was used as input for testing the IDS.

C. Attack data set

In order to test the detection capabilities of each model, we produced another data set, consisting solely of HTTP attack requests, using the compilation from [7][3]. This compilation consists of 63 attacks from different sources: the BugTraq/SecurityFocus archives, the Open Source Vulnerability Database, the Packetstorm archives, and Sourcebank. It contains the following categories of attacks: buffer overflow, input validation error (other than buffer overflow), signed interpretation of unsigned value, and URL decoding error.

[3]Available at http://www.i-pi.com/HTTP-attacks-JoCN-2006.

IV. IDS AND NORMAL TRAFFIC MODELS

The IDS prototype does essentially two things: training and detection. During the *training phase*, it determines the characteristics of requests and creates a statistical model describing normal behavior. In the *detection phase*, requests are tested against the model in order to determine the likelihood of them being attacks. In this phase, the system tests a request and returns a *similarity* value, $s \in [0, 1]$, which represents the similarity of the request being tested to what is expected from the trained model. Values of $s = 0$ indicate a request that was not observed nor derived from the training data. $s = 1$ indicates a request seen during training.

Both the training and detection logic were implemented by us. However, the construction of the models and verification of deviations were done using *IDS::Test*, an IDS test framework developed by K. Ingham [8][4].

A. Models

The *IDS::Test* framework supports several types of models, from which we used nine: length, Mahalanobis distance, χ^2 of idealized character distribution, ordering of parameters, presence or absence of parameters, token-finder, Markov model, combination, and *N*-grams. The IDS uses any of the types of models individually or a combination of the 6 models used by Kruegel and Vigna [9]. We do not have space to present all these models, so we present only those that provided better results.

The *length* of an attribute often can be used to detect anomalous requests [9]. Since parameters are usually either fixed-size tokens or short strings obtained from user input, the length of parameter values should not vary much between requests for the same web application. However, when malicious input is passed to the web application inside parameters, the length of the values is likely to be different from that of normal requests. Length models aim at approximating the parameter lengths and detecting instances that significantly deviate from the observed normal behavior. During the training phase, the mean and the variance of attribute length strings are measured. As for the detection phase, using Chebyshev's inequality, the system calculates the probability that an attribute would have the observed length.

The *Mahalanobis distance* is a standard distance metric used to compare two statistical distributions, which provides a useful way to measure the similarity between the (unknown) new payload sample and the previously computed model. Wang and Stolfo [16] computed the distance between the byte distributions of the newly observed payload against the profile from the model computed for the corresponding length range. The higher the distance score, the more likely the payload is abnormal. We implemented this model following [16] and [6].

[4]Available from CPAN at http://search.cpan.org/~ingham/IDS-Test-1.00/

An *n-gram* [3] is a substring of a string that we get by sliding a window of length n across text. For example, given the text *abcdef* and $n = 4$, the resulting 4-grams are: *abcd*, *bcde*, *cdef*. In order to build a *n*-gram set, it is necessary to break every string into n-grams and store every new *n*-gram into the set. The detection phase for this model simply checks if the *n*-grams in question were observed during training, that is, if they are present in the set of *n*-grams learned from the training data.

B. Generalization

In the training phase, to achieve more than simply memorizing the training data, an anomaly-based IDS has to *generalize*, i.e., to model the data set in a way that is more generic that the specific behaviors in the data set itself. By generalizing, an anomaly-based IDS accepts input similar, but not necessarily identical to that of the training data set, which implies the set of instances considered normal will encompass a larger set than those in the training data. An anomaly detection system that under-generalizes generates too many false positives, while one that over-generalizes can miss attacks. Therefore, correct generalization is a necessary condition for the accuracy of an anomaly-based IDS.

Several heuristics were employed in the detection system in order to increase the generalization for specific areas in the request. After analyzing our data sets we assessed which areas of an HTTP request were plausible for applying generalization, and then decided to use several heuristics. We can not present them all for lack of space, but some examples follow. (1) Instead of learning every IP address and hostname in the Internet, the system validates whether the form of the IP address or the hostname meets the Internet standard. (2) HTTP headers contain hashes, for example, as entity tags or sessions IDs in cookies (e.g., Content-MD5 or PHPSESSID); since the purpose of a hash is to be unique, it does not make sense for an anomaly-based IDS to learn every hash, so the system just validates the form (character set and length) of the hash and returns if the hash is valid or not. (3) Dates can be a problem for an IDS trying to learn the structure of an HTTP request, as they change every day. Thus, the system simply validates whether the date in question meets the RFC 2616.

V. Evaluation

A. Space requirements

One of the requirements for an IDS is that is causes a small overhead in the system it monitors. One metric of such overhead is the amount of memory required to store the model. Table II presents a comparison of size of the models in bytes, when trained with the filtered and the unfiltered data sets. The n-grams are the models which require more space, with the 7-grams requiring 27MB when trained with the filtered data set, and 93MB when trained with the unfiltered data set. As for the remaining algorithms, the Markov model is the one that takes more space, 1.9MB with the filtered data set and 6MB with the unfiltered data set. The other models all take little space.

Model	Filtered data set	Unfiltered data set
3-grams	4,306,864	13,277,971
4-grams	8,053,974	25,294,592
5-grams	13,344,542	42,723,133
6-grams	20,097,675	66,762,546
7-grams	28,293,212	96,863,131
Token	2,164	2,609
Combination	41,004	220,444
χ^2ICD	3,327	3,350
Length	198	200
Mahalanobis distance	7,896	8,198
Markov model	1,915,325	6,011,577
Order	61,124	144,317

Table II
SIZE OF EACH MODEL TRAINED WITH THE FILTERED AND UNFILTERED DATA SETS (IN BYTES).

B. Quality of the IDS

In this section, we evaluate the quality of the IDSs based on the different models by using ROC curves.

When an IDS generates an intrusion alert for a request, this is a *positive*. If the alert refers to a real intrusion attempt, then it is a *true positive*, otherwise it is a *false positive*. On the other hand, if the IDS does not generate an alert, this is called a *negative*. If the IDS is right about it, the alert is a *true negative*, otherwise it is a *false negative*. The *true positive rate* (TPR) or *detection rate* of an IDS measures the amount of malicious events correctly classified and reported as alerts. It is given by $TPR = TP/(TP + FN)$, where TP is the number of true positives and FN is the number of false negatives. The perfect IDS has $TPR = 1$. The *false positive rate* measures the amount of legitimate events that are incorrectly classified and reported as alerts and is given by $FPR = FP/(FP + TN)$, where FP is the number of false positives and TN is the number of true negatives.

Receiver Operating Characteristic (ROC) curves are often used to evaluate the quality of an IDS [12], [4]. The purpose of a ROC curve is to depict graphically the *accuracy* of a detector of some kind. The ROC curve for an IDS is basically a plot between the FPR and the TPR rates as the threshold value is varied, in order to show the tradeoff between them. It is obtained by tuning the IDS to tradeoff false positives against true positives, i.e., each point of the ROC curve corresponds to a fixed amount of TPR and FPR calculated under certain sensitivity parameters (threshold).

To produce the ROC plots, first we ran the IDS in detection mode using as input a set of data that contained only legitimate (normal) requests, and from there we obtained the FPR. Then another set of data, that we knew to contain only attack requests, was used as input for running the IDS in detection mode, and from there we obtained the TPR.

We do not present all the ROCs due to lack of space, only those of the models that provided the best results and one

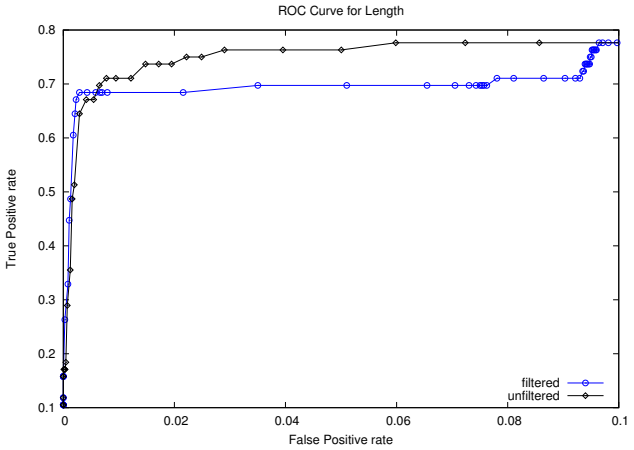

Figure 2. ROC curves illustrating the accuracy of the Length model on the filtered and unfiltered data sets.

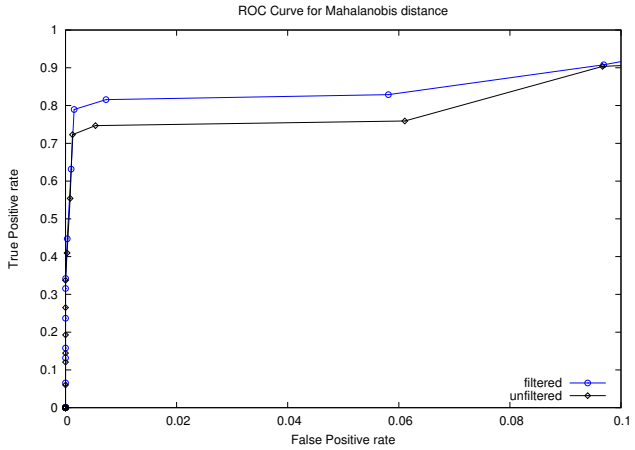

Figure 3. ROC curves illustrating the accuracy of the Mahalanobis distance model on the filtered and unfiltered data sets.

containing all curves.

1) Length: The results for the Length model are presented in Figure 2. When trained with the filtered data set, the detection accuracy for the Length model was considerably worse than when trained with the unfiltered data set, but in either case, it was always below 80%. Regardless, we have observed that using the Length model alone, particularly with a full HTTP request containing headers has some limitations as there are many headers that can influence the final lengths of the request. We were able to improve the accuracy of the Length model with the generalization heuristics mentioned in Section IV-B.

2) Mahalanobis Distance: Figure 3 shows the results of testing the Mahalanobis distance model. As expected, when trained with the filtered data set, this model provided more accurate results than when trained with the unfiltered data set, which was exactly what we aimed for when performing the filtering techniques described earlier. Nevertheless, the best this model could perform was around a 90% true positive rate while maintaining a false positive rate under 10%, which in our opinion is still not sufficient for a production environment.

3) N-*grams:* Figures 4 and 5 show the results of testing the *n*-grams model for $n = 3, 7$. Note the range of the x-axis, i.e., the scale of the FPR, has been changed from 10% to 1%, in order to better visualize the plots. This makes little difference in terms of comparison with the remaining models as all *n*-gram variations achieve a true positive rate of 100% before the false positive rate reaches 1%.

Against our expectations, for all the *n*-grams variations, the detection was more accurate when the unfiltered data set was used for training, rather than when the filtered data set was used, despite the differences being almost insignificant to even mention. The *3*-grams model was also the most accurate model when it came to testing, but in general, for any of the five different *n*-grams models that were used, the

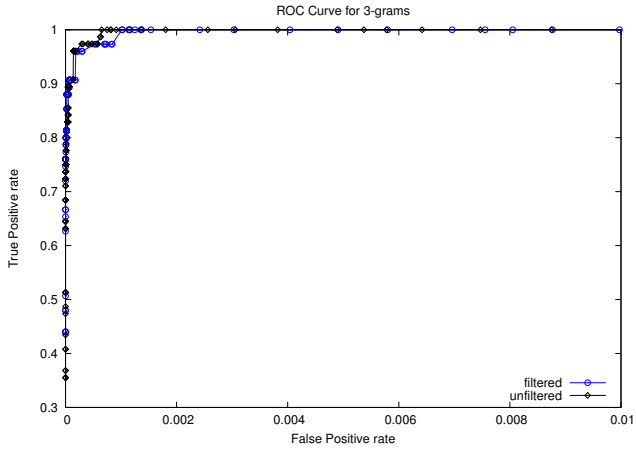

Figure 4. ROC curves illustrating the accuracy of *3*-grams on the filtered and unfiltered data sets.

detection rate was always very high and the false positive rate almost nonexistent. However, it was surprising that the smaller the *n*, the better this model performed. The reason for this lies in the fact that most malicious parameters involve rarely seen characters rather than a different combination of usually seen characters, which can also explain why it drastically outperforms the Length model.

In our experiments, *n*-gram modeling of web requests shown to be a very promising and accurate model to detect attacks, with a very high detection rate and very low false positive rate. Clearly, this was the most appropriate model for detecting attacks with the real-world data we used to build our models and train our system.

4) Model comparison: Ordering models by accuracy depends on the acceptable false positive rate, but in generally the ROC curves of the models are well-separated throughout the graph. In order to depict the difference in terms of accuracy between the models, Figure 6 presents a comparison between the 3-grams, 7-grams, Mahalanobis distance, Length,

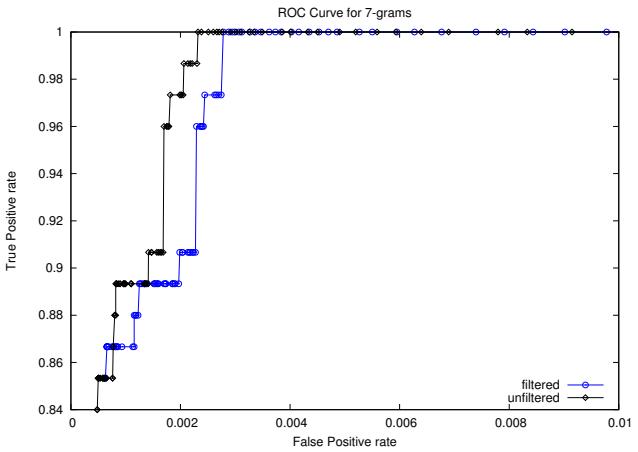

Figure 5. ROC curves illustrating the accuracy of 7-grams on the filtered and unfiltered data sets.

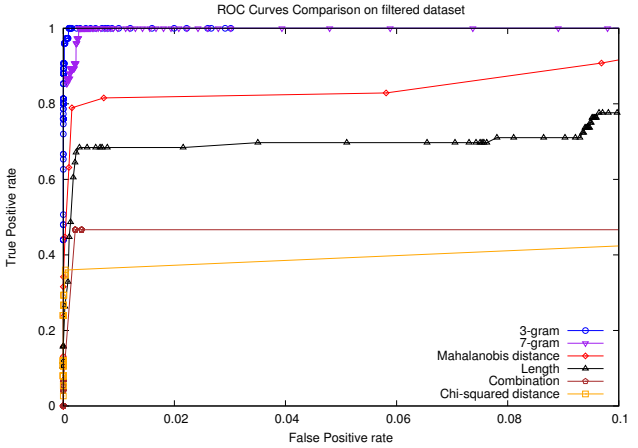

Figure 6. ROC curves comparing the accuracy of the 3-grams, 7-grams, Mahalanobis distance, Length, Combination and χ^2 distance when trained with the filtered data set.

Combination and χ^2 distance, when using the filtered data set to construct the models. The *n*-grams models are clearly the ones that show the highest true positive and lowest false positive rates. For the remaining models, the Mahalanobis distance is the one that performs better, followed by the Length. There is however a considerable gap between the n-grams and Mahalanobis distance and the remaining ones, Length, χ^2 distance and Combination.

VI. Conclusion

Anomaly-based intrusion detection is a promising technique since it allows detecting previously unknown attacks, which is important as new vulnerabilities and attacks are constantly appearing. The paper presented a study of anomaly-based intrusion detection with a large SaaS application. It presents how data was obtained and sanitized. A comparison of the several models that can be used to represent normal behavior has shown that n-grams provide the best accuracy, with a high detection rate and a low false positive rate.

Acknowledgments

This work was partially supported by Fundacão para a Ciência e a Tecnologia through project RC-Clouds (PCT/EIA-EIA/115211/2009) and the Multiannual and CMU-Portugal Programmes. This work was done while the authors were with the Universidade de Lisboa, Faculdade de Ciências and the Carnegie Mellon University Information Networking Institute.

References

[1] E. G. Amoroso. *Intrusion Detection*. Intrusion.Net Books, 1999.

[2] S. Cho and S. Cha. SAD: web session anomaly detection based on parameter estimation. *Computers & Security*, 23(4):312–319, 2004.

[3] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[4] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[5] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. CSI/FBI computer crime and security survey. *Computer Security Institute*, 25, 2005.

[6] K. L. Ingham. *Anomaly detection for HTTP intrusion detection: algorithm comparisons and the effect of generalization on accuracy*. PhD thesis, University of New Mexico, 2007.

[7] K. L. Ingham and H. Inoue. Comparing anomaly detection techniques for HTTP. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, pages 42–62. Springer, 2007.

[8] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

[9] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 251–261, 2003.

[10] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5):717–738, 2005.

[11] A. Luotonen. The common log file format. http://www.w3.org /pub/www/, 1995.

[12] R. A. Maxion and R. R. Roberts. Proper use of ROC curves in intrusion/anomaly detection. Technical Report CS-TR-871, Newcastle University, 2004.

[13] W. Robertson, G. Vigna, C. Kruegel, R.A. Kemmerer, et al. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proceedings of the 13th Symposium on Network and Distributed System Security*, February 2006.

[14] M. Roesch et al. Snort-lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, pages 229–238, November 1999.

[15] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.

[16] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the 7th International Conference on Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.