A Proposal of Algorithm for Web Applications Cyber Attack Detection

Rafal Kozik, Michal Choraś, Rafal Renk, Witold Holubowicz

A proposal of algorithm for web applications cyber attack detection

Rafał Kozik[1,2], Michał Choraś[1,2], Rafał Renk[1,3], Witold Hołubowicz[2,3]

[1] ITTI Ltd., Poznań, Poland mchoras@itti.com.pl
[2] Institute of Telecommunications, UT&LS Bydgoszcz, Poland
rafal.kozik@utp.edu.pl
[3] Adam Mickiewicz University, UAM, Poznan, Poland renk@amu.edu.pl

Abstract. Injection attacks (e.g. XSS or SQL) are ranked at the first place in world-wide lists (e.g. MITRE and OWASP). These types of attacks can be easily obfuscated. Therefore it is difficult or even impossible to provide a reliable signature for firewalls that will detect such attacks. In this paper, we have proposed an innovative method for modelling the normal behaviour of web applications. The model is based on information obtained from HTTP requests generated by a client to a web server. We have evaluated our method on CSIC 2010 HTTP Dataset achieving satisfactory results.

Keywords: web attacks detection, web applications firewall, machine learning, data mining

1 Introduction

Currently, providing effective cyber-security solutions for web applications is very challenging. This happens due to the fact that the commonly used IDS (Intrusion Detection System) and IPS (Intrusion Prevention System) systems have problems in recognising new attacks (0-day exploits), since these systems are based on the signature-based approach. In such a mode, when the system does not have an attack signature in its database, such attack is not detected. Therefore, there is a need to develop more sophisticated methods that are both capable of adapting domain expert knowledge [1][2] as well as emerging cyber security solutions (e.g. event correlation [3] and data mining).

The list of top 10 most critical risks related to web applications security, provided by OWASP (Open Web Application Security Project [4]) indicates "Injection" vulnerabilities as the major vulnerability. The Injection flaws, such as SQL, OS, and LDAP injection occur when improperly validated data containing malicious code is sent to an interpreter as part of a command or query.

According to the OWASP ranking, the second on the list are attacks related to Broken Authentication and Session Management. Incorrectly implemented authentication usually allows attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users identities.

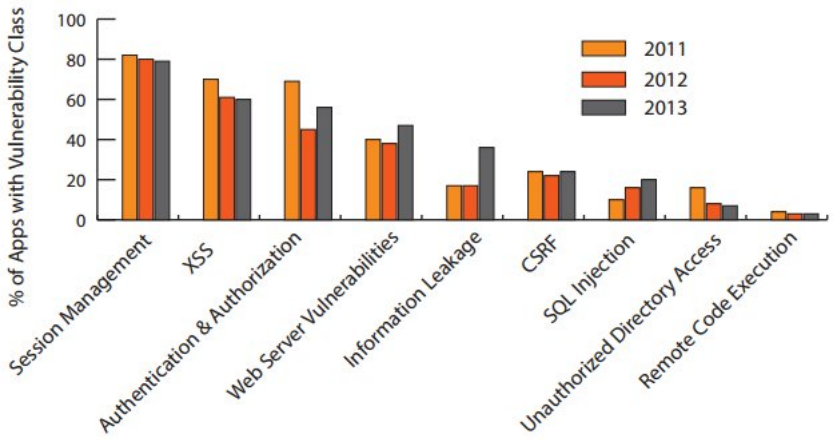

Fig. 1. The 2013 vs. 2012 and 2011 Web Application Vulnerabilities Trends(source [5])

According to Application Vulnerability Trends Report [5] the session management and authorization vulnerabilities are one of the most frequently identified problems during the last three years (2010-2013).

The XSS (Cross Site Scripting) take the third place on the OWASP list. The XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without a proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Therefore, in order to counter those top ranked problems,in this paper we have proposed an innovative method for modelling the normal behaviour of web application. The model is based on information obtained from HTTP requests generated by client to a web server.

The remainder of this paper is structured as follows. First, we give an overview of methods for we application attacks detection. Next, the detailed method description is presented. The experiments set-ups as well as the results are presented in following section. Conclusions are given thereafter.

2 Overview of methods for web application attacks detection

There are several tools and methods for detecting the cyber attacks targeting web applications.

Some of the frequently used tools use static code analysis approaches in order to find the vulnerabilities that may be exploited by any cyber attack. Some examples of such tools include PhpMiner II [6], STRANGER [7], AMNESIA [8]. However, as it is stated in [9], the difficulty relates to the fact that many

kinds of security vulnerabilities are hard to find automatically (e.g. access control issues, authentication problems). Therefore, currently such tools are only able to automatically find a relatively small fraction of application security flaws.

One of the most commonly used and popular class of tools for web application cyber attacks detection adapts signature-based approach to describe (and detect) cyber attacks. Some examples include PHP-IDS [10], SCALP [11], Snort [12]. The biggest advantage of such tools is their ability to process huge amounts of data. This is due to the fact that there are efficient algorithms that are able to check a given piece of text against a pattern (usually expressed as PCRE [13] regular expressions) in a short time. However, the common drawback is that an expert knowledge is required to build such patters describing cyber attack. Moreover, such attacks like SQL injection are easy to obfuscate (e.g. using URL encoding). This makes the problem of providing a reliable pattern of an attack very difficult.

3 Proposed method overview

The proposed method overview is shown in Fig.2. It adapts a machine-learning paradigm, therefore two distinct phases are presented on the diagram. During the learning phase, the labelled data is required in order to establish the model parameters of normal application behaviour. As mentioned before, only HTTP request headers are used for model training and these need to be labelled as either normal or anomalous.

As it is shown in Fig.2 A, the HTTP requests need to be parsed in order to extract significant parts. In this approach, first the URL (e.g. https://host/users) is extracted and concatenated with the HTTP method name (e.g. GET, POST, PUT, etc.). Those two parameters are used to create a key (address) to entry in hashmap. During the learning phase, the hashmap is populated only with a normal (legitimate) HTTP request. This allows us to build a whitelist of resources that are usually requested by users via HTTP protocol. Whenever the HTTP request has parameters (e.g. parameter1=value1&parameter2=value2), it is encoded using the method described in section 3.1. However, the method produces vectors that are of different length. Therefore, to make it possible to learn a classifier, we transform this vector to histograms of constant length. The final feature vector is extended with information whenever a given request is on the whitelist or not.

During the testing phase (or when the algorithm operates in a production environment), the key is established using the same procedure as before (URL is concatenated with the HTTP method). If the HTTP request contains the parameters, it is encoded in order to produce the feature vector that is extended with information whenever a given request is on the whitelist or not. Such feature vector is recognized as normal or anomalous with the classifier learnt before.

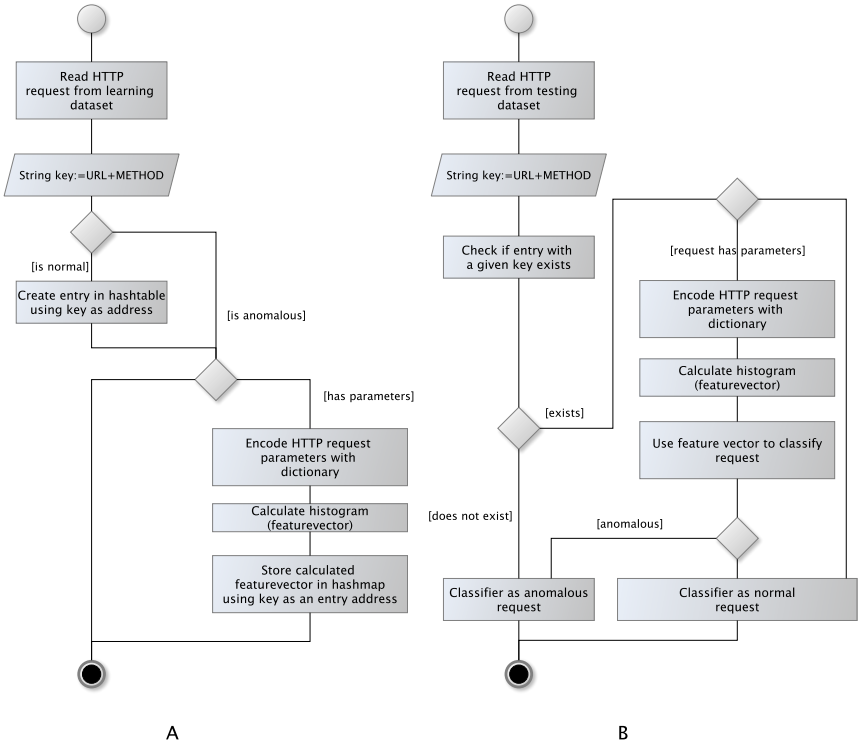

Fig. 2. The proposed algorithm (A indicates learning phase, while B indicates testing/classification phase)

3.1 Encoding HTTP parameters

In order to encode the HTTP parameters as a feature vector, we use dictionary D that maps parts of text to a set of natural numbers (see equation (1)).

$$D : word \rightarrow \{i : i \in \mathbb{N}\} \tag{1}$$

The dictionary is established on a learning set using the algorithm (1) that adapts a modification of LZW compression method (Lempel-Ziv-Welch [14]). In contrast to the original LZW method we first establish the dictionary D during the learning phase and then encode the text (we do not extend the dictionary further when the method operates in the production environment).

The algorithm scans through the input request parameters S for successively longer substrings until it finds one that is not in the dictionary. If a given substring in not in the dictionary, then it is added and the whole procedure is repeated until the whole learning data set is processed.

Data: Set of HTTP request parameters S
Result: Dictionary D
s = empty string
while *there is still data to be read in* S **do**
  $ch \leftarrow$ read a character;
  if $(s + ch) \in D$ **then**
    $s \leftarrow s+ch$;
  else
    $D \leftarrow D \cup (s + ch)$;
    $s \leftarrow ch$;
  end
end

Algorithm 1: Algorithm for establishing dictionary D

Once the dictionary D is established, the HTTP request parameters are encoded. The algorithm scans again through the input request parameters S for successively longer substrings until it finds one that is not in the dictionary. The longest substring is encoded with the natural number that indicates its position in the dictionary D. The procedure is described by algorithm (2).

Data: Set of HTTP request parameters S, dictionary D
Result: Vector V of natural numbers
s = empty string
while *there is still data to be read in* S **do**
  $ch \leftarrow$ read a character;
  $V \leftarrow 0$;
  if $(s + ch) \in D$ **then**
    $s \leftarrow s+ch$;
  else
    $V \leftarrow V \cup D(s)$;
    $s \leftarrow ch$;
  end
end

Algorithm 2: Algorithm for encoding text with dictionary D

4 Experimental set-up

For the experiment the CSIC'10 dataset [15] was used. It contains several thousands of HTTP protocol requests which are organised in a form similar to Apache Access Log. The dataset was developed at the Information Security Institute of CSIC (Spanish Research National Council) and it contains the generated traffic targeted to an e-Commerce web application. For convenience the data was split into anomalous, training, and normal sets. There are over 36000 normal and 25000 anomalous requests. The anomalous requests refer to a wide range of application layer attacks, such as: SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, XSS, and parameter tampering.

Moreover, the requests targeting hidden (or unavailable) resources are also considered as anomalies. Some examples from this group of anomalies include client requests for: configuration files, default files or session ID in URL (symptoms of HTTP session take over attempt). What is more, the requests not having the appropriate format (e.g. telephone number composed of letters) are also considered anomalous. As the authors of the dataset explained, such requests may not have a malicious intention, but they do not follow the normal behaviour of the web application.

According to the authors knowledge, there is no other publicly available dataset for the web attack detection problem. The datasets like DARPA or KDD'99 are outdated and do not cover many of the current attacks.

5 Results

For evaluation purposes we have adapted the 10-fold cross-validation technique.

For that approach, the data obtained for learning and evaluation purposes is divided randomly into 10 parts (sets). For each part it is intended to preserve the proportions of labels (number of anomalies and normal feature vectors) in the full dataset. One part (10% of full dataset) is used for evaluation while the remaining 90% is used for training (e.g. establishing model parameters).

When the classifier is learnt the evaluation data set is used to calculate the error rates. The whole procedure is repeated 10 times, so each time different part is used for evaluation and different part of data set is used for training.

The result for all 10 runs (10-folds) are averaged to yield an overall error estimate.

In these experiments we have evaluated such classifiers as: J48, PART, AdaBoost, and NaiveBayes.

Table 1. Effectiveness for CSIC 2010 HTTP Dataset.

	Detection Rate (True Positives)	False Positive Rate
Nguyen et al. [16]	93.65%	6.9%
NaiveBayes	88.89%	6.26%
AdaBoost	83.23%	15.45%
PART	93.35%	2.79%
J48	**95.97%**	**3.54%**

In Tab.1, a comparison of different classifiers is presented. Moreover, we reported the effectiveness of our method using as a baseline the approach proposed by Nguyen et al. in [16] (the authors of CSIC 2010 HTTP Dataset).

The ROC curve for different classifiers has been presented in Fig.3. It was generated with the WEKA tool [17], which varies the threshold on the class probability estimates. The best results have been observed for J48 tree classifier. It was possible to achieve a better detection rate while having lower false positive

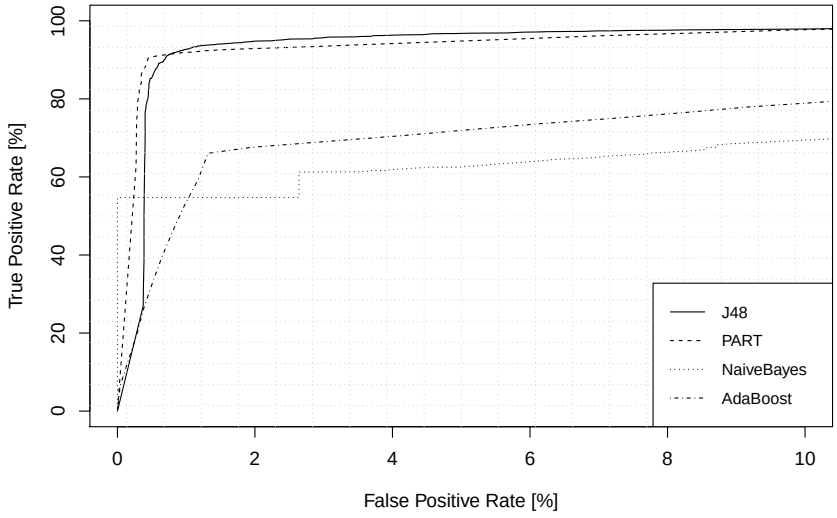

Fig. 3. ROC curve for different classifiers.

rate in comparison to the method proposed by Nguyen et al. in [16]. Additional, we have used t-test to evaluate the statistical significance of the obtained results. Test showed that the results are statistically significant at 0.95 level.

6 Conclusions

In this paper we have proposed an innovative method for detecting current cyber attacks targeting web applications.

We have particularly focused on solutions that are using HTTP protocol to communicate clients with the servers. We have shown that recent cyber incidents reports prove that there is an increasing number of attacks targeting these web-based applications.

The analysis show that the attacks exploiting injection vulnerabilities are still one of the most dangerous and frequently reported by institutions gathering statistics about the network incidents.

The proposed algorithm for detecting the cyber attacks targeting the web applications relies on fact that it is more effective to model normal behaviour of an application (observing the HTTP traffic) than to produce the reliable attack signature.

We have evaluated the proposed method using CSIC 2010 HTTP Dataset. The experiments have shown that the proposed method achieves satisfactory results. Moreover, we have compared our method with method proposed by

CSIC Dataset authors. We report that our method that it is able to achieve the higher detection rate while having lower false positive rate.

References

1. Choraś M., Kozik R., Flizikowski A., Holubowicz W., Ontology Applied in Decision Support System for Critical Infrastructures Protection, In N. Garcia-Pedrajas et al. (Eds.): Trends in Applied Intelligent Systems, IEA/AIE 2010, Part I, LNAI 6096, pp. 671-680, 2010.
2. Choraś M., Kozik R., Piotrowski R., Brzostek J., Holubowicz W., Network Events Correlation for Federated Networks Protection System, In: Abramowicz W. et al. (Eds.): Towards a Service-Based Internet, LNCS 6994, pp. 100-111, Springer 2011
3. Choraś M., Kozik R., Network Event Correlation and Semantic Reasoning for Federated Networks Protection System, In Chaki N. et al. (Eds.): Computer Information Systems - Analysis and Technologies, Communications in Computer and Information Science CCIS, 48-54, Springer, 2011.
4. OWASP Top 10 2010, The Ten Most Critical Web Application Security Risks. (2010)
5. Application Vulnerability Trends Report `http://www.cenzic.com/downloads/Cenzic_Vulnerability_Report_2014.pdf`
6. Shar, LwinKhin, and HeeBengKuan Tan. "Predicting common web application vulnerabilities from input validation and sanitization code patterns." Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 2012
7. Yu, Fang, MuathAlkhalaf, and TevfikBultan. "Stranger: An automata-based string analysis tool for PHP." Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2010
8. Halfond, William GJ, and Alessandro Orso. "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, 2005.
9. Source Code Analysis Tools. `https://www.owasp.org/index.php/Source_Code_Analysis_Tools`
10. PHP-IDS project homepage. `https://phpids.org/`
11. Apache Scalp Project homepage. `http://code.google.com/p/apache-scalp/`
12. Snort project homepage. `http://www.snort.org/`
13. Perl-compatible regular expressions (pcre) `http://www.pcre.org`
14. LZW algorithm. http://en.wikipedia.org/wiki/LempelZivWelch
15. CSIC 2010 HTTP Dataset. `http://users.aber.ac.uk/pds7/csic_dataset/csic2010http.html`
16. Nguyen H., Torrano-Gimenez C., lvarez G., Petrovic S., Franke K. Application of the Generic Feature Selection Measure in Detection of Web Attacks. In Proc. of International Workshop in Computational Intelligence in Security for Information Systems (CISIS 11), LNCS 6694, pp. 25-32, 2011.
17. WEKA tool. ROC curve generation. `http://weka.wikispaces.com/ROC+curves`