

WEB APPLICATION FIREWALL

CS499 : B.Tech Project Final Report

by

Namit Gupta (Y3188)
Abakash Saikia (Y3349)

under the supervision of

Dr. Dheeraj Sanghi

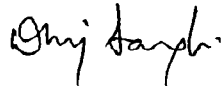


submitted to

Department of Computer Science and Engineering
Indian Institute Of Technology, Kanpur

Certificate

Certified that the work titled "Web Application Firewall" by Namit Gupta and Abakash Saikia has been done under my supervision and the work has not been submitted elsewhere for a degree.



Dr. Dheeraj Sanghi

30/4/07

Abstract

Today applications are becoming the prime target for cyber attacks. A recent research showed that approximately 70% of all successful web attacks exploit application vulnerabilities and there is no shortage of vulnerabilities to go after, all of them require some skill to exploit. While traditional firewalls have blocked packets effectively at the network layer, they are ineffective against attacks which point to application weaknesses. Web application firewalls detect application deviation and whether sensitive data, such as account information or credit card number, is being hacked and can take suitable action accordingly. Moreover, Intrusions pose a serious threat to the computer world and must be dealt with correctly. Everyday some new kind of attack comes into existence and pose a threat to the application security. Current signature based methods and machine learning algorithms cannot detect such intrusions as they rely on training of labeled data.

We divided the implementation of the Web Application Firewall into five different parts. When a HTTP request comes, it is initially parsed into useful chunks of information, which are then normalized to some standard format. Some built-in-checks are performed on the normalized form which is followed by checking of user-defined rules. Logging is done intelligently on every request. Unsupervised learning techniques over unlabeled data are used to detect different types of intrusions while maintaining a low false positive rate. Using heuristics and a learning engine, an effort towards averting Zero Day Attacks is made.

Introduction

A Web Application Firewall (WAF) is a security module in an application proxy device that protects the web application server in the back end from various attacks. Application protection is a valuable security layer to add because it can protect against a number of application layer security threats which is usually not protected by a typical network layer intrusion detection system. The WAF ensures that the security of web server it is protecting is not compromised by looking at the HTTP/HTTPS request packets (Deep packet inspection) and the web traffic patterns. On finding any kind of security threat with accordance to the configuration file or by the intrusion detection system, the WAF prevents the attack by blocking the HTTP request or user session or by IP address.

Logging forms a major part of any web application. At times, it becomes so very important to log things as they help in detecting some flaw or activity by some malicious user at a later point of time. Presently, in most of the softwares log analysis is not being done intelligently, the information is just put into the logs and later handled manually. But, it becomes a waste when we have a large number of requests coming to our application and useful chunks of information can be extracted from these requests. Many new attacks can be detected and hence the backend server must be provided with better security. The Network Intrusion Detection System (IDS) tries to detect such attacks by analysing the data and trying to find suspicious patterns.

Generally, the algorithms used in the IDS use various techniques to detect intrusions. For e.g. Signature based methods use hard-coded algorithms given by some experts to detect already known attacks. Data mining techniques also use labeled data to train. But, these algorithms fail when it comes to detecting new kind of intrusions, which are not yet known. To counter it, either a new algorithm has to be given or the system has to be trained again

on the new dataset.

Our method is based on the concept of anomaly detection using clustering. Anomalies are something which behave differently from the normal data. Clustering techniques work on the fact that similar data instances be combined into a cluster and the anomalies be defined based on their distances from these clusters. Two assumptions are made for this method to work - normal instances have the same properties which are quite different from those of anomalies and the number of normal instances must be exceedingly large than the intrusions. So, after clustering of the data, clusters with less number of elements can be considered as intrusions.

Related Work

The Web Application Security Consortium (WASC) is an international group of experts which has been looking at the best-practice security standard for the World Wide Web. There are many other groups working in this area, Thinking Stone being one of them, which has its open source product called Mod Security. It is an open source WAF acting as a module to an apache web server. Currently, it is the most widely deployed WAF. It can avert already known attacks with the help of configuration files. New attacks cannot be detected. So, the attacks like the Zero Day Attack are not taken into account. We have tried to take these extra features into our product.

As far as intrusion detection goes, a lot of clustering algorithms have been studied already. Y-Means technique initially initializes k clusters randomly. Then the elements are clustered based on the least distance from the cluster center. If there is some empty cluster remaining, then we replace those empty clusters. Instances are then labeled according to the population and k is then adjusted by splitting or merging clusters. It gives greater accuracy than other techniques but is quite costly in terms of time which is an important factor in intrusion detection systems. Other algorithms use probability distribution, statistics, decision trees etc. to detect anomalies in the data. Some of them have worked fine while others lack in some or the other aspect. The algorithms we have used are Leonid Portnoy algorithm, which tries to detect anomalies by measuring euclidean distances and K-Means, an algorithm similar to Y-Means.

Technical Details

Deployment Scenario

WAF sits between the insecure internet and the web server. The request from a client to the backend server is analyzed at the WAF. The safe requests are then sent to the server while the malicious ones are being dropped there itself. WAF would be very generic irrespective of the backend server which can be either a database or an office workstation. It can be shown as a context diagram in the form as given in Figure 1.

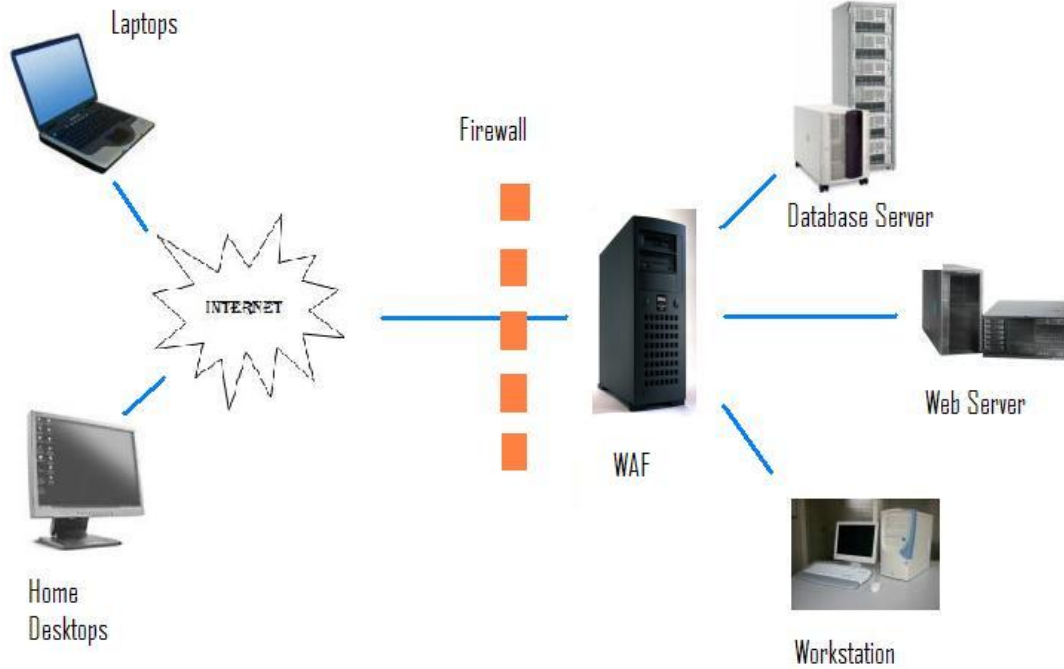


Figure 1: Deployment Scenario

Implementation Details

The design of the WAF divided into individual components goes as follows:

(a) *Parsing of Request*

A HTTP request can be said to be composed of different parts all of which occurring in a certain order. So, a normal HTTP request is seen as:

$$\begin{aligned}
 Request \Rightarrow & RequestLine \\
 & * ((GeneralHeader \\
 & |RequestHeader \\
 & |EntityHeader)CRLF) \\
 & CRLF \\
 & [MessageBody]
 \end{aligned}$$

The request line is of the form of:

$$Request\ Line \Rightarrow Method\ SP\ Request\ URI\ SP\ HTTP\ version$$

The methods can be one from GET, PUT, POST, CONNECT, HEAD etc. A header can be said to be supplemented data at the beginning of a packet and contains information like remote user, authorization type, host, time, arguments, cookies, etc. Message body is optional and comes after the header part.

(b) *Normalization of Request*

This part deals with converting the requests to a standard format. Request from different sources are generally in different forms and hence are converted to a standard general form. This step fights against various evasive techniques such as null byte attacks, self-referencing directories, multiple slash characters, URL encoding etc. An example of self referencing directories can be:

We have set a rule which matches against the string “/bin/sh”. But a user can request for the file as “/bin/./sh” which is the same path as the first one but our rule fails to match against the request string and the given file can be accessed by the user. Thus, such requests need to be normalized to some standard format.

This is basically divided into two major parts - one for URL encoding and the other for checking threats like self referencing directories, multiple slash characters etc. As shown in Figure 2, function other is responsible for normalizing self referencing directories, multiple slash characters and Null byte attacks. Apart from it, there is a function for URL Encoding normalization, which calls functions detect_unicode_character and filter_multibyte.

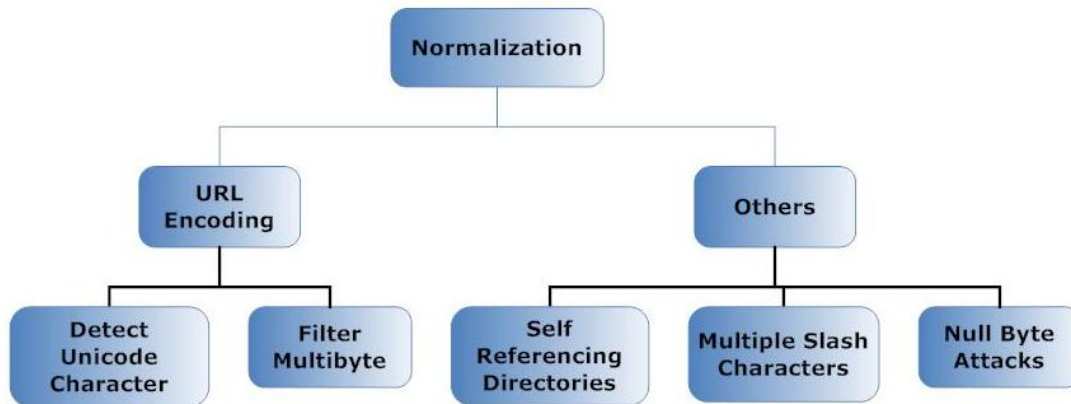


Figure 2: Normalization methods

(c) *Basic built-in checks*

This part helps in making some basic security checks built into the system irrespective of different configurations. These are basically hard coded into the module. This would include complicated validations such URL encoding validation and Unicode encoding validation.

Special characters need to be encoded before they can be transmitted in the URL. Any character can be replaced using the three character combination %XY, where XY represents an hexadecimal character code. Hexadecimal numbers only allow letters A to F, but

attackers sometimes use other letters in order to trick the decoding algorithm.

(d) *Execute matching of rules*

The rules are expressed as Regular Expressions. Each rule represents a certain type of threat which the administrator thinks to be vulnerable to the system. Each rule is compared with the input string i.e. information received from the packet. In case of matching, the packet is blocked by closing the socket connection or else it moves to the next rule. For e.g.

```
Filter REMOTE_ADDR ‘‘203.200.95.130’’ | REQUEST_FILENAME ‘‘/bin/sh’’  
‘‘deny’’
```

The above rule filters the request seeing the IP address of the client and the file requested by him/her. So, if the IP equals 203.200.95.130 or the requested filename is “/bin/sh”, then it will simply block the packet.

(e) *Logging*

Logging forms an important part of any web application. Here, logs are maintained both sides, from client to server and back from server to client. A total of 18 fields like protocol, source/destination IP address, source/destination port number, method, response status, time, etc is being logged. Thereafter, log file is being trained using the clustering algorithms which helps in detecting an intrusion into the system.

Process Flow

A normal HTTP packet when it arrives is first stopped at the proxy server and given to the firewall. The information in the packet is being checked against the basic rules in the configuration file. If it is considered to be a safe packet, then it is given to the backend server otherwise the connection is closed right there. The server responds in a normal fashion and provides the client with the required page. Logs are maintained both ways, on the arrival of a packet and on way back too. The flow can be shown in the form of a diagram as shown in Figure 3.

Clustering Algorithms

The clustering algorithms use the dataset to form clusters and detect intrusions. Almost all of them work on the following two assumptions:

1. The normal instances have similar properties and occur close together to form one single cluster while anomalies are far apart from them.
2. The number of normal instances are exceedingly large than the number of anomalies i.e. normal instances constitute around 95-98 % of the total data while anomalies constitute the rest.

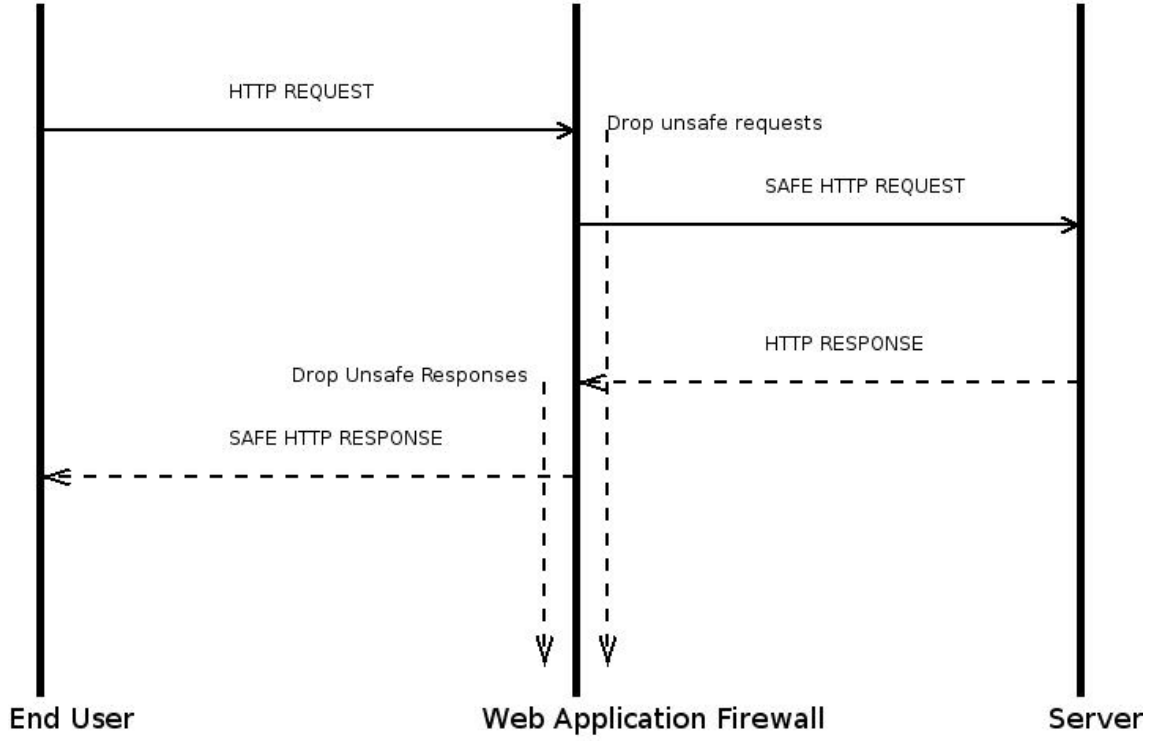


Figure 3: Process Flow

The next two algorithms which we are going to discuss next have been tested on the KDD Cup 1999 dataset. It consists of around 5 million data instances, each of which is a 41-dimensional vector obtained from raw network data. The fields are duration, protocol type, number of bytes transferred, flag indicating normal or error status of the connection etc. Each connection was labeled as either normal or as exactly one kind of attack.

As system designed must work for instances coming from an arbitrary distribution, data instances must be normalized in order to form clusters. The normalization can be done by calculating the average and standard deviation vectors as:

$$avg[j] = \frac{1}{N} \sum_{i=1}^N data_i[j]$$

$$std_dev[j] = \left(\frac{1}{N-1} \sum_{i=1}^N (data_i[j] - avg[j])^2 \right)^{1/2}$$

New data instance is then given as:

$$new_data[j] = \frac{data[j] - avg[j]}{std_dev[j]}$$

The euclidean distance is given by the root of the squared distance between two feature

vectors.

$$euc_dist = \sqrt{\sum_{i=1}^N (a_i - b_i)^2}$$

Leonid Portnoy Algorithm

We use a simple variant of single-linkage clustering. The algorithm starts with empty set of clusters. For each new instance, it computes the distance between it and the centers of clusters formed so far. The cluster with the shortest distance is calculated and if it is less than some fixed distance W , it is assigned to that cluster. Otherwise, a new cluster is made with it being the center. It can be given as:

1. Initialize the set of clusters, S , to the empty-set.
2. Get a data instance from the dataset. If S is empty, then this data instance becomes the defining distance and add it to S . Otherwise, find the cluster whose center is closest to it.
3. If this distance is smaller than the the cluster width W , then add the instance to that cluster. Otherwise, create a new cluster for it and add the cluster to S .
4. Repeat steps 2 and 3 until no instances are left in the set.

When applied on the KDD-99 dataset, Leonid Portnoy gives a result of around 65%.

K-Means

K-Means is a typical clustering algorithm. It partitions the dataset into k clusters according to the following steps:

1. Choose k instances randomly from the dataset and make them initial centers for the clustering space.
2. Take each instance from the dataset and assign it to the closest cluster.
3. Replace each center with the mean of its members.
4. Repeat steps 2 and 3 until there is no more updating of the center.

The difference it has with the Leonid Portnoy algorithm is choosing of k , the initial number of clusters, in the first case and W , the cluster width, in the latter one. When applied on the KDD-99 dataset, K-means gives a result of around 80%.

As already stated, our dataset consists of a set of 18 vectors which we get from the

contents of a HTTP packet. Some of the vectors are protocol, source IP address, destination IP address, source port number, destination port number, method, host, payload size, response status and time.

In our assumption, we have mentioned that a large number of instances are normal while the rest are anomalies. So, we label some percentage N of the clusters containing the largest number of instances as normal, rest being anomalies. After having trained the dataset by our algorithms, a new instance is put into some cluster using the formulae and the cluster is checked against a normal or an anomalous one. A voting mechanism is used and if both the algorithms classify the new instance as an anomaly, then only we consider it as an anomaly otherwise not.

Integration

We have to integrate the Web Application Firewall with the proxy server so that the packets which are coming from the client side can actually be stopped on matching of rules. For this, we have used an open source proxy server, Wcol. It is a multi process proxy system and can handle many connections at the same time.

The basic working mechanism of Wcol integrated with WAF is:

A socket connection is setup between the client and the proxy server. Whenever a client wishes to access some page from the backend server, it issues a GET request to the proxy server. For each accepted socket connection, a new instance of method *Reception()* is forked. So, we may have more than one processes running at the same instant. The packet information is then stored in a structure and rule matching is done. If it comes out to be a safe packet, the proxy server then fetches the page from the backend server and sends it back to the client. Otherwise, the connection is closed there itself and the client is denied access.

Zero Day Attack

A zero-day attack is a computer threat that exposes unpatched computer application vulnerabilities. Zero-day attacks can be considered extremely dangerous because they take advantage of computer security holes for which no solution is currently available. Zero-day attacks can occur because a vulnerability window exists between the time a threat is released and the time patches are released. So, it is very essential to find out such vulnerabilities as soon as possible.

Any attacker discovering the vulnerability would try to exploit it early before the patch releases. So, basically packets of similar kind which try to exploit the vulnerability will increase abruptly. This property of abruptness in increase of such packets can be exploited to find out Zero Day attacks. Our intrusion detection system finds out intrusion packets on the basis of clustering. Clusters with less number of elements than some threshold are considered intrusion. So, any abrupt increase in number of elements in an intrusion cluster can be marked as Zero Day Attack possibility and the administrator be informed so that more detailed check on vulnerabilities against such packets be made.

The basic issue is the detection of abruptness detection. We implemented a short time average versus long time average threshold detector, which computes two exponentially

weighted moving averages with same gain constants. It applies weighting factors which decrease exponentially. This makes sure it gives more importance to recent observations while still not discarding older observations completely. The degree of weighting decrease is termed as smoothing factor. Smoothing factor at a time period N is given as:

$$Smoothing_Factor = \frac{2}{N + 1}$$

And the Exponentially Weighted Moving Average (EWMA) is given by:

$$EWMA_t = Smoothing_factor * Y_t + (1 - Smoothing_factor) * EWMA_t - 1$$

When the ratio between the short time average and the long term average exceeds a certain threshold, the detector informs that this may be a Zero Day Attack.

Results

Finally, we were able to implement a Web Application Firewall on a proxy server. It allows or blocks HTTP traffic based on the rules mentioned in the configuration file. Apart from it, Artificial Intelligence based intrusion detection system is implemented on the WAF which divides the packets into clusters and later categorizes them into normal packets or intrusion. Moreover, a new Zero Day Attack detection technique was also implemented by calculating the abruptness of increase in the intrusion packets.

Acknowledgements

Apart from our efforts, the success of this project depends largely on the encouragement and motivation of many others.

We would like to show our greatest appreciation to Dr. Dheeraj Sanghi. Without his guidance and support, this project would not have been materialized. A special mention to the Juniper people whose guidance was also vital for the success of the project. We are grateful for their support and help.

References

- [1] Ivan Ristic et al., “Web Application Firewall Evaluation Criteria”, Web Application Security Consortium, 2006, <http://www.webappsec.org/projects/wafec/>.
- [2] Mod Security, Thinking Stone Limited, 2006, <http://www.modsecurity.org/>.
- [3] Jeffrey R. Williams et al., “The Ten Most Critical Web Application Security Vulnerabilities”, The Open Web Application Security Project, 2004, http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
- [4] Leonid Portnoy, “Intrusion Detection with unlabeled data using clustering”, Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001), Philadelphia, PA: November 5-8, 2001.
- [5] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy and Salvatore Stolfo, “A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data” Data Mining for Security Applications, Kluwer 2002.
- [6] Guan Y., Ghorbani A., and Belacel N., “Y-Means: A Clustering Method for Intrusion Detection”, Canadian Conference on Electrical and Computer Engineering, May 2003.
- [7] Guan Y., Ghorbani A., and Belacel N., “K-Means+: An Autonomous Clustering Algorithm”, Technical Report TR04-164, University of New Brunswick, 2004.
- [8] Tapas Kunango, Nathan S. Netanyahu and Angela Y. Wu, “An Efficient K-Means Clustering Algorithm: Analysis and Implementation”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 7, July 2002.