CSE508 – Network Security

Web Application Firewall
Project Report

Team:

Devashish Thakur, SBU ID: 109770589
Naveen Kumar Nuthalapati, SBU ID: 109746917
Kaushik Devarajaiah, SBU ID: 109889142
Karthik Reddy, SBU ID: 109721778

Table of Contents

I. Introduction

A web application firewall (WAF) is a system running in-front of a web server that is responsible for inspecting all traffic coming from the Internet and relaying it to the actual web server. If WAF sees traffic that contains an attack, it will just drop the request and not forward it to the application. WAFs have the ability to protect vulnerable web applications by ensuring that malicious traffic never reaches the vulnerable code.

In this project, we have designed and implemented a Web Application Firewall that runs in-front of a web server as an Apache Module. WAF is able to recognize malicious requests using two methods: *signatures* and *anomaly detection*.

II. Design Considerations

- **Modular** - Implemented signature based filtering and profile based filtering as two different modules.
- **Dynamic** - WAF is able to generate normality profiles either on the fly during request handling or offline.
- **Homogeneous technology usage** - everything is coded in C (even the code that generates normality profiles).

III. Design

We chose to implement WAF as an Apache module. It is configured to be the first module to be invoked on every incoming HTTP request to the web server.

As shown in the figure below, all incoming requests are redirected to WAF module. WAF module inspects the request in two stages based on signatures and normality profiles. If the request is found to be malicious in any of the stages, WAF simply drops the request without sending it to the webserver and sends back *HTTP 404 Not Found* response to the client. Otherwise, the request is forwarded to the web application and appropriate response is sent back to the client.

We have developed a web application using WordPress and PHP.

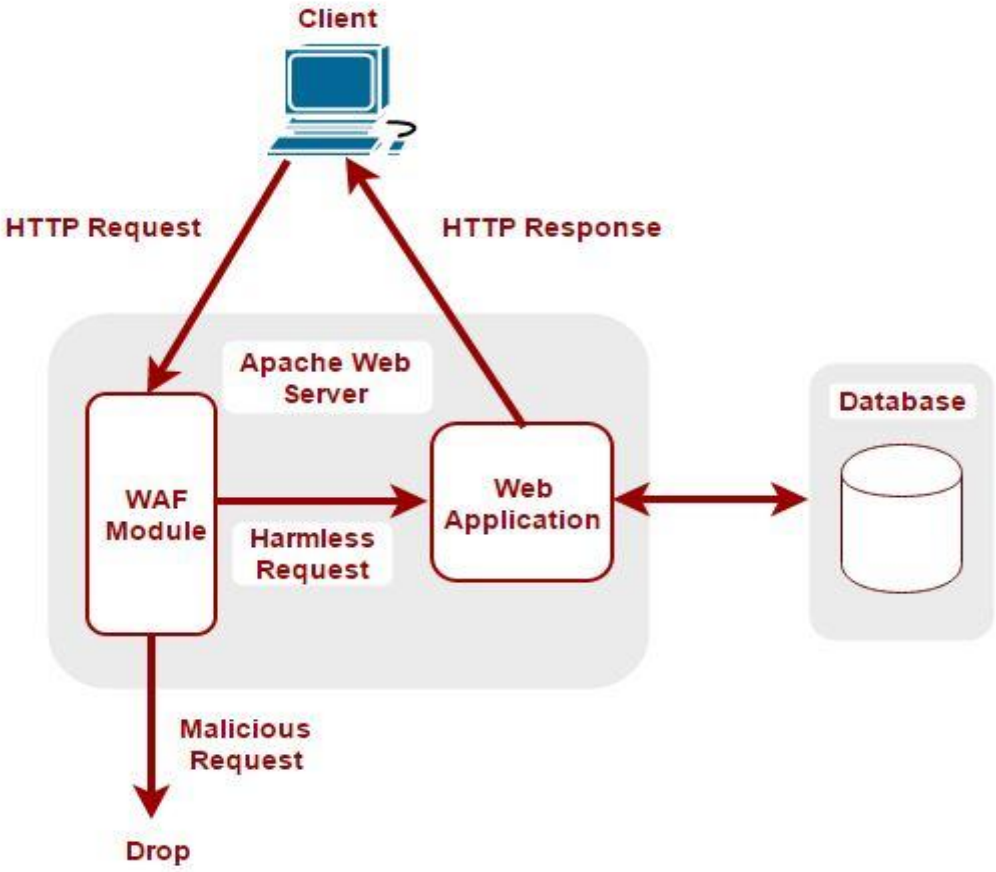

Fig 1: Web Application Firewall (WAF) Design

IV. WAF in detail

Following are the sequence of steps that a request undergoes inside a WAF module.

1. Browser sends an HTTP request to Apache web server and WAF module is invoked.
    a. Every request information is appended to the log file.
    b. The WAF will look through all the information in the logs and try to make generalized rules called normality profile which is used in profile based filtering. This could be done for every request or be generated offline and support for both exist.
2. Signature based filtering – In this stage, WAF parses signatures file and regex matching is applied to perform signature based filtering.
3. Profile based filtering – WAF reads the generalized rules in the normality profile generated in step 1b. Based on these rules, it checks whether the request is allowed or not. If the request is found to be violating any of the rules, WAF drops the request. Otherwise, the request is forwarded to the web application.

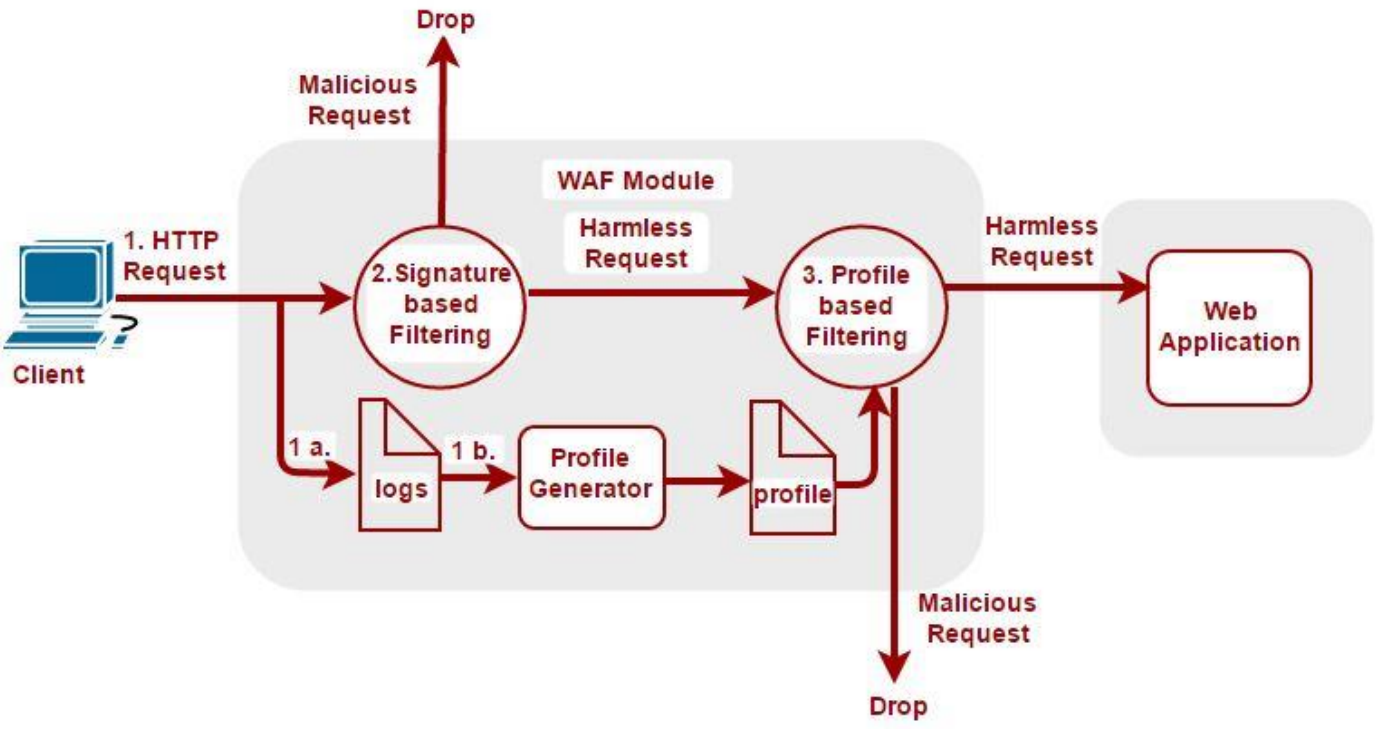

Fig 2: Web Application Firewall (WAF) in detail

V. Detection Methods

1) Signature based filtering

There are 3 core functions in this module.

a. **parseSignatures** - Parses the given signatures file.
The signature file has the following format:

```
HEADER:User-Agent,CONTAINS:"<script>"
HEADER:User-Agent,CONTAINS:"bot"
REQUEST_METHOD:GET,PARAMETER:*,CONTAINS:"union all select"
REQUEST_METHOD:POST,PARAMETER:foo,CONTAINS:"../../../../"
```

b. **headerAllow** - Takes a header field's value as an input, then applies regex matching on the header value. The regex checks if the header contains any of the phrases from the signatures file. If there is a match, request is dropped.
An example input to this function would be:

Example 1:

```
"User-Agent###Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101
Firefox/21.0"
```

If the signatures file contained any of the following entries, the request would be dropped.

```
HEADER:User-Agent,CONTAINS:"Gecko"
HEADER:User-Agent,CONTAINS:"mozilla"
```

Example 2:

```
"Accept-Encoding###gzip, deflate"
```

If the signatures file contained any of the following entries, the request would be dropped.

```
HEADER:User-Agent,CONTAINS:"gzip"
HEADER:User-Agent,CONTAINS:"deflate"
```

c. **req_allow -** Takes an input string of format " <REQUEST-METHOD> <URL>". The URL is of type "/<resource-path>?<key>=<value>&<key>=<value>". Using regex matching, the function then checks each of the key and value pairs for a match in the signatures file. If there is a match, then the request is dropped.
An example input to this function would be:

```
"GET /index.html?foo=bar&par=union all select"
```

If the signatures file contained any of the following entries, this request would be dropped.

```
REQUEST_METHOD:GET,PARAMETER:*,CONTAINS:"union all select"
REQUEST_METHOD:POST,PARAMETER:foo,CONTAINS:"bar"
```

2) Profile based filtering

While signatures are a great way to stop known attacks, they can't stop attacks that do not have corresponding signatures. In profile based filtering, we learn how legitimate traffic looks like and then if some future traffic is too different from what we've learned, we do not allow it to proceed. Here, the system is split into a training phase and an online phase.

In the training phase, the WAF does not stop anything. It just collects information about all traffic passing through it. The user is then trying to use the website as much as possible, click on all possible links, and perform all possible actions. When the user is satisfied that he/she has exercised the web application as much as possible, user asks the WAF to create a normality profile. The WAF will look through all requests and try to make generalized rules. In this project, we made WAF to learn following things:

- Maximum number of parameters seen for all requests across all pages.
- Maximum number of parameters seen for every specific page.
- Average length of values for every specific parameter of every specific page.
- Character set of any specific parameter.

This module has two core methods and the detection happens as follows:

a. **store_data()** - This generates the profile by reading the logs.
b. **is_url_valid(url)** - This checks the validity of the url based on the generated profile. It happens as follows:
    - For profiles involving maximum number of parameters, requests that exceed the maximum number are dropped.
    - For profiles involving averages, a normal distribution is assumed, standard deviation is computed, and all values that are not part of the mean+-3*sd are dropped.
    - For profiles involving character sets, requests with parameters containing character from sets not seen during training are dropped.

VI. Technologies Used

- C
- Apache Web Server
- apxs (apache module builder)
- PHP
- WordPress
- MySQL
- Libraries – regex.h, math.h

VII. Conclusion

Web Application Firewall (WAF) analyses requests to the webserver at application layer on HTTP protocol. It is designed and developed as a configurable apache module. It is configured as the first module to be invoked on every incoming request. It handles black list based filtering of requests based on signatures. It is capable of generating normality profiles either offline or during request handling and filters requests which greatly deviate from normality profile.

VIII. Task Division

Karthik Reddy – Signature based filtering, integration with profile based filtering.
Devashish Thakur – Profile generation and profile based request filtering.
Naveen Kumar Nuthalapati – PHP applications, WordPress setup and configuration, Project Report, Presentation.
Kaushik Devarajaiah – Exploration of Apache modules, interface definition and code integration, installation of Apache, PHP, MySQL.

IX. References

1. http://www.linuxuser.co.uk/tutorials/develop-apache-modules/3
2. http://httpd.apache.org/docs/2.2/install.html
3. http://php.net/manual/en/install.unix.apache2.php
4. https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-14-04
5. https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-on-ubuntu-14-04