Addressing Application Layer Attacks with Mod Security

This article sheds some light on some of the important concepts pertaining to Web Application Firewalls (WAF). We have also looked at the Mod_Security Apache module as an example of WAF. Here, we would discuss the detail implementation of Mod Security Apache Module, while configuring connectivity between Apache Webserver and Tomcat container. We would also look at the installation of Mod_Security module on Apache.

What is a Web Application Firewall (WAF)?

In simple terms, a Web Application Firewall (WAF) can be defined as an appliance, server module or a filter that imposes a set of rules to HTTP traffic. The rules include signatures to detect some of the common attacks, such as Cross Site Scripting (XSS), SQL Injections and Parameter Manipulation Attack.

WAF is an intermediary device, which sits in front of Webserver and filters out layer 7 traffic. WAF could be a software or hardware appliance that can be installed in an environment with little or no infrastructural changes. WAF can provide protection to Webservers even if the Web application has no in-built secure code.

Why WAF?

Or even a better one: "Why do I need a WAF, when I already have IDS and IPS in place?" Well, the answer is quite obvious - IDS and IPS provide passive detection, but WAF provides active detection and prevention. In the following sections, we see how active detection and prevention can be achieved with Mod Security.

Mod Security

Mod Security is a WAF solution for Apache Webservers. Basically, Mod Security is a module for Apache Webserver (Note: It is not an Apache Tomcat Webserver) and it needs to be configured with Apache HTTP Server. Mod Security provides the framework where Web administrators can create rules to monitor and restrict the HTTP traffic.

Steps to connect Apache with Tomcat

Dynamic pages like Servlets and JSP cannot be hosted over Apache Webserver. Tomcat Webservers should be used to run Servlets or JSP pages. It's useless to protect an Apache HTTP server, which doesn't host any dynamic (JSP or Servlet) page.

There are some good techniques to connect Apache HTTP server and Tomcat server. The idea is to listen to HTTP or static pages on Apache server and redirect JSP or Servlet requests to Tomcat server. It has already enhanced a layer of security by putting more than one server between the Tomcat and the end-user.

Performing a DOS attack on the Webserver leads to a strike or a hit on the Apache HTTP Server.

Even if the Apache server goes down, it would still be easy in bringing up the Website as the Apache server was only hosting static pages. Having the Apache server before Tomcat server in fact helps in fine-tuning the performance as both the servers share the load.

The steps to connect Apache Webserver and Tomcat are listed below:

JKconnector is used to connect Apache with tomcat. JKconnector can be downloaded from the Apache site. The main purpose of it is to get JSP and servlet to run on port 80 rather than setting Tomcat to run on port 80. Thus, connector acts as a conduit between the two.

Step 1: The downloaded file mod_jk must be put in the \Apache2.2\modules folder. The downloaded file should be renamed as mod_jk.

Step 2: Edit httpd.conf located in \Apache2.2\conf. This is done to load the jk connector at the run time with some of the following commands:

```
<IfModule !mod_jk.c>

        LoadModule jk_module modules/mod_jk.so

        // tells Apache to load the mod_jk module
```

```
</IfModule>
```

Step 3: Edit tomcat configuration file (Server.xml) and add following code just below

```
< Server port="8005" shutdown="SHUTDOWN" debug="0">

<Listener className="org.apache.jk.config.ApacheConfig"

modJk="C:/Program Files/Apache Software

Foundation/Apache2.2/modules/mod_jk.so" />
```

Next, look for the code below in the server.xml file:

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"

xmlValidation="false" xmlNamespaceAware="false" debug="0">
```

Once the above parameters are found, add the following code below it:

```
<Listener className="org.apache.jk.config.ApacheConfig"

append="true" forwardAll="false" modJk="C:/Program Files/Apache

Software Foundation/Apache2.2/modules/mod_jk.so" />
```

By adding the two listener elements in the server.xml, we are making the Tomcat automatically generate the necessary Apache configuration directives for Mod_jk. Now we don't have to generate those directives manually.

Step 4: After saving the changes made in the server.xml, restart the Tomcat services. Subsequently, check for a file called mod_jk.conf in the \Tomcat 5.5\conf\auto.

Step 5: There is a concept of worker in Apache module to Send and Receive information to the Tomcat. Hence, we need to create workers.properties file, which has location information of the Tomcat and the port to be used. We are putting this workers.properties file in \Apache2.2\conf. Accordingly, the path of workers.properties file has to be specified in the httpd.conf as mentioned in step 3 commands.

Following lines have to be present in the workers.properties file.

```
worker.list=ajp13

worker.ajp13.port=8009

worker.ajp13.host=localhost

worker.ajp13.type=ajp13
```

These entries define a Tomcat worker name ajp13 that resides on the same host as the Apache server, localhost listens to port 8009 for a client using the ajp13 protocol (it is a packet based protocol that allows a Webserver to communicate with the Tomcat jsp/servlet container over a tcp connection and has a better support for ssl).

Step 6: Again edit httpd.conf file located in \Apache2.2\conf and put below code right after the code we have entered in step 2.

```
<IfModule !mod_jk.c>

        LoadModule jk_module modules/mod_jk.so

        // tells Apache to load the mod_jk module

                JkWorkersFile "conf/workers.properties"

                //tells the location of properties file

                JkLogFile "logs/mod_jk.log"

                JkLogLevel error

                JkMount /jsp-examples ajp13
```

```
            JkMount /jsp-examples/* ajp13

            JkMount /Application ajp13

            JkMount /Application/* ajp13

</IfModule>
```

Jkmount directive /Application/* suggests Apache that all requests to be rerouted and serviced by the worker named ajp13. One can also specify patterns instead of * (all). For example /Application/.jsp will allow only .jsp pages to be run on the Apache server.

The Tomcat server needs to be started first followed by the Apache Webserver. Application should be hosted on the Tomcat server (which is running on port 80) and the request would be sent to Apache and then redirected to the Tomcat server.

Steps to Install modsecurity

To install the modesecurity, create …/apache2/modules/mod_security2 folder and copy mod_security2.so and libxml2.dll to this folder.

Add to httpd.conf file:

```
LoadModule security2_module modules/mod_security2/mod_security2.so
```

Configuring WAF

After having restarted Apache, mod_security will be installed but not activated and enabled. In order to activate mod_security, we need to add some directives to httpd.conf file. Here are few examples to get started:

```
#Enable mod_security module

#Possible values ON/OFF/DynamicOnly

SecFilterEngine ON



#Retrieve request payload

#Possible values ON/OFF

SecFilterScanPost ON



#Verify the encoding of URL request

#Possible values ON/OFF

SecFilterCheckURLEncoding ON



#Enables logging Functions

#Possible values ON/OFF/RelevantOnly

#ON - All HTTP requests will be logged
```

```
#OFF - Nothing will be logged at all

#RelevantOnly - Requests matches any filter will be logged only

SecAuditEngine ON



#Used in conjunction with SecAuditEngine and

#specifies the location of the log file.

#Possible values – File path

SecAuditLog /log/log_file.log



#Allows scanning of POST request.By default only GET requests are scanned

#Possible values ON/OFF

SecFilterScanPOST



#Default action when a filter is matched. In below example, mod_security

#will reject all invalid requests with status 403 and log this action

#Possible values : Discussed in later part of the article

SecFilterDefaultAction "deny,log,status:403"



#Disallows directory traversal
```

```
SecFilter "\.\./"


#Deny all request containing /bin/bash string

#Possible values ON/OFF

SecFilter /bin/bash


#Allows to set debug log path

#Possible values – Debug log file path

SecFilterDebugLog /log/mod_debug_log


#Disallows directory traversal

#Possible values 1 and 4

#1 for production and 4 for testing

SecFilterDebugLevel 4
```

Processing Order

The Mod_security module will first look for the SecFilterEngine directive. Only if this directive is set to ON, will the mod_security module continue processing. Similarly, if SecFilterEngine is set to DynamicOnly and current request is for static resource, processing will be terminate immediately.

Now If SecFilterEngine is ON, mod_security initializes its structure. Mod-security reads the entire body part line by line and checks requests against rules configured in the module. If any part of the request fails validation, mod_security directly jumps to the default action (as set in the SecFilterDefaultAction directive.) For example, if a rule is configured to accept only numeric values then any attempt to send nonnumeric value will be rejected and default action will come in place.

Defining Your Own Rules

The best part of mod_security is its Rule engine. The module allows users to easily define and implement rules. Rules needs to be inserted in httpd.conf file as mentioned earlier. One can define a rule by a single keyword. The SecFilter directive performs a broad search against the request parameters.

Here is the Syntax for the SecFilter directive:

SecFilter Keyword

This keyword can be as simple as 100 or Select. The module will look for any occurrence of specified keyword in entire HTTP request; in this case 100 or Select.

To optimally use mod_security, use regular expressions to make search patterns effective and accurate. With the help of regular expression we can have filter on ranges, digits, wildcards or on combination. An example of regular expression is ^(|[0-9]{1,4})$ which will look for all numeric entries between 1 and 4 digits in length.

Broad rules are easy to write but they become a little complex at times. The wide use of broad rules eventually results in false-positive responses and sometime restricts legitimate users to make use of the system.

In order to fine tune your search pattern to create a powerful and more effective rule base, ModSecurity provides many advanced options. One of these is the SecFilterSelective directive. This directive allows to filter on CGI variables, such as QUERY_STRING, REMOTE_ADDR, PATH_INFO, REQUEST_URI, and TIME_YEAR. See ModSecurity documentation for complete list of CGI variables. The use of arguments is also supported in ModSecurity. Rules can be specified to check against arguments carried in any HTTP request. We will take an example to understand this feature in the later part of the article.

Available Actions

As seen in the earlier part of the article, there are a number of actions available in ModSecurity, a few of them are:

Allow: Allows matching requests through.

Deny: Stops the request outright, returns a HTTP 500 error code by default.

Status: Used to specify an alternate HTTP error code.

Redirect: Matching requests are redirected to the provided URL.

Log: Logs request only.

Nolog: Does not log request.

Chain: Allows you to create list of filters.

Now let's take a look at couple of examples to understand the SecFilterSelective directive:

```
SecFilterSelective REMOTE_ADDR "^192.168.53.$" log, chain

SecFilterSelective Request_URI "index\.htm" \

Redirect:http://test.com/home.htm
```

In above example we have created chain of two rules. First rule will check whether request is coming from 192.168.53.x subnet. If yes then the rule engine will pass control to the second rule. The Second rule checks to see what file is requested. If the requested file is index.htm, the user will redirect to an alternate location (home.htm).

```
SecFilterSelective ARG_username "^admin$" chain

SecFilterSelective Remote_ADDR "^192.168.52.4&"
```

In this example, first rule looks for the argument username carried in HTTP request. If the username starts and ends with admin then the second rule will be looked at. The second rule allows the request to be further analyzed if its destination address is 192.168.52.4.

Conclusion

Web Application Firewalls are an excellent addition to secure web servers from application layer attacks. A well configured WAF can protect web servers from almost all kind of application layer attacks including XSS, SQL Injection, and Parameter Manipulation attacks. With its ability to filter on any HTTP, HTTPS GET and POST request, we can address all malicious requests targeting our web servers. Support of Regex (Regular Expressions) allows security admin to create more powerful rule base.

WAF is not intended to take place of Firewall, IDS or IPS, it has an ability to filter HTTP traffic. Overall WAF is a great concept to secure web servers from Layer 7 attacks and it can secure badly coded applications to some extent.