



<http://www.modsecurity.org>

Copyright (c) 2002-2004 Ivan Ristic

ModSecurity for Java (prototype 1)

Introduction

This document contains various notes written during my work on the Java version of ModSecurity. For more information about ModSecurity, please take a look at the web site and the documentation for the Apache version. My intent at the moment is to make the Java version fully compatible with the Apache version (in terms of configuration).

Feel free to send me comments, ideas, requests. My email address is ivanr@webkreator.com

This version is **prototype 1**. I want to play with classes and design of the software, refactoring as I go. Please do not expect that future versions will be backwards compatible (plugin architecture, for example).

Installation

Quick start

To have a quick look around:

1. Drop modsecurity-p1.war into your webapps folder
2. Open <http://www.yourwebserver.com/modsecurity-p1/> in a browser

The first page should show links to three files:

- You should be able to access the first file normally.
- The access to the second file will be denied because its name contains a keyword "123".
- The access to the third file will be denied because its name contains a keyword "xxx". However, this event will also trigger a full IP address ban for the duration of 30 seconds. You can verify this by trying to access the first file, which is not protected.

Standard configuration

Add the file modsecurity-prototype1.jar to the application classpath.

ModSecurity for Java is a Servlet filter, written according to the v2.3 spec. The software was

tested with Tomcat 5 but should work on any 2.3 compatible container. Here is how the file web.xml looks like:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>ModSecurity/Java Prototype 1 WAR</display-name>

  <filter>
    <filter-name>modsecurityFilter</filter-name>
    <filter-class>com.webkreator.modsecurity.ModSecurity</filter-class>
    <init-param>
      <param-name>conf</param-name>
      <param-value>WEB-INF/modsec.conf</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>modsecurityFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

The engine is configured with a configuration file modsec.conf. The syntax is the same as the syntax of the Apache Web server (it will stay that way, too). Only two configuration directives are implemented at the moment:

- **SecFilter** – reject access if a keyword is present in the URI
- **SecRudeWordFilter** – reject access if a keyword is present, plus reject all access from that IP address for the next thirty seconds..

Both filters look for keywords in an URI as returned by method `HttpServletRequest.getRequestURI()`.

Creating a plugin

ModSecurity plugins implement the Plugin interface. This is how an empty plugin looks like:

```
class EmptyPlugin implements Plugin {

  public boolean init(SecurityContext securityContext) {}

  public void shutdown() {}

  public int perform(HttpServletRequest req, HttpServletResponse res) {
    return Plugin.CONTINUE;
  }
}
```

You can see that a plugin does its job when the method `perform()` is executed. The plugin then needs to perform its checks, and return one of the three available constants:

- Plugin.CONTINUE – continue with the request as usual
- Plugin.STOP – interrupt request processing and perform the default action
- Plugin.QUIT – interrupt request processing immediately, performing no actions (assumes the plugin has performed what is required)

Plugin lifecycle

One plugin instance is created per security context. The lifecycle of an instance is:

- init method is called
- perform method is called zero or more times
- shutdown method is called

Configuration directives

Each plugin can register a number of configuration directives with ModSecurity. Directive is a class implementing interface Directive. One approach a plugin writer could take is to make the plugin implement both interfaces, like this:

```
class SimplePlugin implements Plugin, Directive {

    public boolean init(SecurityContext securityContext) {
        securityContext.registerDirective("SimpleDirective", this);
    }

    public void shutdown() {}

    public String handleDirective(String[] tokens) {
        System.out.println("SimpleDirective found in configuration.");
    }

    public int perform(HttpServletRequest req, HttpServletResponse res) {
        return Plugin.CONTINUE;
    }
}
```

The first token is always a String instance containing the name of the directive. Therefore, a single handleDirective can handle any number of directives.

Plugin communication

Plugins can communicate with other plugins that are in the same context. Every plugin that is loaded is registered with the context, using its full class name as key. This way some plugins can expose their services to other plugins. For example, ModSecurity currently implements IpControlPlugin, which can reject requests based on IP address (indefinitely, or for a limited time only). Like this:

```
// find the plugin instance
IpControlPlugin ipControl = (IpControlPlugin)securityContext.findPlugin(
    "com.webkreator.modsecurity.IpControlPlugin");
```

```
if (ipControl != null) {  
    // ban this IP address for 30 seconds  
    ipControl.banIp(request.getRemoteAddr(), 30000);  
}
```

Note: it is also likely that plugins will be able to issue configuration directives dynamically. For example, the equivalent directive to the code given above would be:

```
SecIpControlBanIP <ip-address> 30000
```

Communication through directives is somewhat safer because you do not depend on a particular plugin class for a certain functionality. The request will go to any plugin registered for the directive. However, this method is also limiting, as the return object from directive execution will always be a String instance.

Complete plugin example

Take a look at the file `GenericRulePlugin.java` for a complete working example.