Tutorial: Configuración de Rate Limit en Apache con ModSecurity y mod_headers

Este tutorial explica cómo implementar un sistema de **rate limit** en Apache con **ModSecurity** y **mod_headers**, que limite las solicitudes a **10 por segundo** bajo las siguientes condiciones:

- Método HTTP: `POST`
- La URL termina en `/acs`
- El cuerpo de la solicitud contiene la palabra `PERIODIC` en una etiqueta `<EventCode>`

Además, se incluirá el encabezado `Retry-After` en las respuestas **429 Too Many Requests**, indicando al cliente cuánto tiempo debe esperar antes de reintentar.

1. Instalar Apache

Si no tienes Apache instalado, puedes hacerlo ejecutando los siguientes comandos:

```
sudo apt update
sudo apt install apache2
```

2. Instalar ModSecurity

Instala **ModSecurity**, que es el módulo que nos permite crear reglas personalizadas de seguridad y control:

```
sudo apt install libapache2-mod-security2
```

3. Habilitar ModSecurity y mod_headers

Asegúrate de que los módulos **ModSecurity** y **mod_headers** están habilitados:

```
sudo a2enmod security2
sudo a2enmod headers
sudo service apache2 restart
```

4. Configurar ModSecurity

Copia el archivo de configuración recomendado de ModSecurity:

```
sudo cp /etc/modsecurity/modsecurity.conf-recommended
/etc/modsecurity/modsecurity.conf
```

Edita el archivo `/etc/modsecurity/modsecurity.conf` para activar el motor de reglas cambiando la siguiente línea:

```
SecRuleEngine DetectionOnly
```

Por:

```
SecRuleEngine On
```

5. Crear las reglas de rate limit

Crea un archivo de configuración para las reglas personalizadas de ModSecurity:

```
sudo nano /etc/apache2/modsecurity-ratelimit.conf
```

Agrega las siguientes reglas al archivo:

```
# Filtro solo si la URL termina en /acs y el método es POST
SecRule REQUEST_URI "@endsWith /acs" "phase:1, id:1001, t:none, pass, nolog"
SecRule REQUEST_METHOD "@streq POST" "phase:1, id:1002, t:none, pass, nolog"

# Verificar que haya exactamente una etiqueta <EventCode>
SecRule REQUEST_BODY "@rx <EventCode>.*?</EventCode>" "phase:2, id:1003, capture,
pass, nolog"
SecRule TX:0 "@eq 1" "phase:2, id:1004, pass, nolog"

# Verificar que el contenido de <EventCode> contenga la palabra 'PERIODIC' sin
importar mayúsculas/minúsculas
SecRule REQUEST_BODY "@rx <EventCode>\s*PERIODIC\s*</EventCode>" "phase:2,
id:1005, t:lowercase, pass, nolog"

# Incrementar el contador global de solicitudes solo si se cumplen todas las
condiciones anteriores
SecAction "id:1006, phase:2, pass, nolog, setvar:global.req_counter=+1,
expirevar:global.req_counter=1"

# Limitar a 10 solicitudes por segundo para todas las solicitudes que cumplan las
condiciones
SecRule GLOBAL:req_counter "@gt 10" "phase:2, id:1007, t:none, deny, status:429,
msg:'Rate limit exceeded', \
    setenv:RATELIMIT_RETRY=1, setvar:'tx.retry_time=5'"

# Añadir el encabezado Retry-After utilizando mod_headers si se supera el límite
de solicitudes
Header always set Retry-After "5" env=RATELIMIT_RETRY
```

Explicación de las reglas:

1. **ModSecurity**:
    - Se utiliza **ModSecurity** para gestionar el límite de solicitudes. Si se superan las 10 solicitudes por segundo, el servidor responde con **429 Too Many Requests** y configura la variable de entorno `RATELIMIT_RETRY=1`.
2. **mod_headers**:
    - El módulo **mod_headers** añade el encabezado `Retry-After` solo cuando se establece la variable de entorno `RATELIMIT_RETRY=1`, lo que asegura que solo se incluya el encabezado cuando se haya alcanzado el límite.
3. `Retry-After`:
    - El encabezado `Retry-After` está configurado para indicar 5 segundos de espera, pero puedes ajustar este valor según tus necesidades cambiando `"5"` en el archivo.

6. Incluir las reglas personalizadas en Apache

Modifica el archivo de configuración principal de Apache `/etc/apache2/apache2.conf` para incluir el archivo de reglas personalizadas que acabas de crear:

```
Include /etc/apache2/modsecurity-ratelimit.conf
```

7. Reiniciar Apache

Reinicia Apache para aplicar los cambios:

```
sudo service apache2 restart
```

8. Probar el rate limit

Usa el siguiente script en Node.js para probar que el **rate limit** está funcionando correctamente y que el servidor responde con **429 Too Many Requests** junto con el encabezado `Retry-After`:

Instalación de `node-fetch`

Primero, instala `node-fetch` si no lo tienes:

```
npm install node-fetch
```

Crear el script de prueba

Crea un archivo `test.js` con el siguiente contenido:

```
const fetch = (...args) => import('node-fetch').then(({ default: fetch }) =>
fetch(...args));

// Configuración del ratio de peticiones por segundo
const ratioPerSecond = 15; // Cambia este valor para ajustar el ratio y probar el
límite

// URL a la que se enviarán las peticiones
const url = 'http://localhost:8080/acs';

// Cuerpo de la petición SOAP que se va a enviar
const soapBody = `
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
    <SOAP-ENV:Body>
      <cwmp:Inform>
        <Event>
          <EventStruct>
            <EventCode>PERIODIC</EventCode>
            <CommandKey></CommandKey>
          </EventStruct>
        </Event>
      </cwmp:Inform>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>`;

// Configuración de la petición POST
const requestOptions = {
  method: 'POST',
  headers: {
    'Content-Type': 'text/xml',
  },
  body: soapBody,
};

// Función que envía una petición
function sendRequest() {
  fetch(url, requestOptions)
    .then(response => response.text())
    .then(result => {
      console.log('Request sent:', result);
    })
    .catch(error => {
      console.error('Error sending request:', error);
    });
}

// Función que controla el envío de las peticiones con el ratio especificado
```

```
function startSendingRequests(ratio) {
  const interval = 1000 / ratio; // Intervalo de tiempo en milisegundos entre cada
petición
  setInterval(sendRequest, interval);
}

// Iniciar el envío de peticiones con el ratio especificado
startSendingRequests(ratioPerSecond);
```

Ejecutar el script

Ejecuta el script para probar las reglas de rate limiting:

```
node test.js
```

El servidor debería limitar las solicitudes a **10 por segundo** y, cuando el límite sea superado, devolver una respuesta **429 Too Many Requests** junto con el encabezado `Retry-After: 5`, indicando que el cliente debe esperar 5 segundos antes de intentar nuevamente.

Este tutorial te guía paso a paso para configurar un sistema de **rate limit** en Apache utilizando **ModSecurity** y **mod_headers**, asegurando que el encabezado `Retry-After` se envíe correctamente en las respuestas cuando el límite es excedido.