



Fuente: Microsoft Copilot

LABORATORIO 10. DEFENDIENDO APLICACIONES WEB

DEPARTAMENTO DE INFORMÁTICA. UNIVERSIDAD DE OVIEDO

Seguridad de Sistemas Informáticos | 2024 – 2025 (v4.2 "S-81 Isaac Peral")





Contenido

⬅ Infraestructura de este Laboratorio	2
📁 Bloque 1: Instalación de un proxy inverso	2
🛠 Ejercicio L10B1_REVPROXY: Apache 2 instalado como un proxy inverso	3
📁 Bloque 2. Web Application Firewalls (WAFs)	5
🛠 Ejercicio L10B2_WAF: Instalación de un WAF	5
🛠 Ejercicio L10B2_WAFPROT: WAF contra ataques web	8
📁 Bloque 3: Herramientas AST para el desarrollo de aplicaciones	9
🛠 Ejercicio L10B3_SASTSCA: <i>Herramientas de detección automatizada de vulnerabilidades en aplicaciones</i> (SAST, SCA)	9
🛠 Ejercicio L10B3_HARDPROXY: Combinación del proxy con técnicas anteriores	13
🛠 Ejercicio L10B3_DAST: Uso de un DAST contra una aplicación	14
💡 Insignias y autoevaluación	18

◀
BACK

1

2

3





BACK

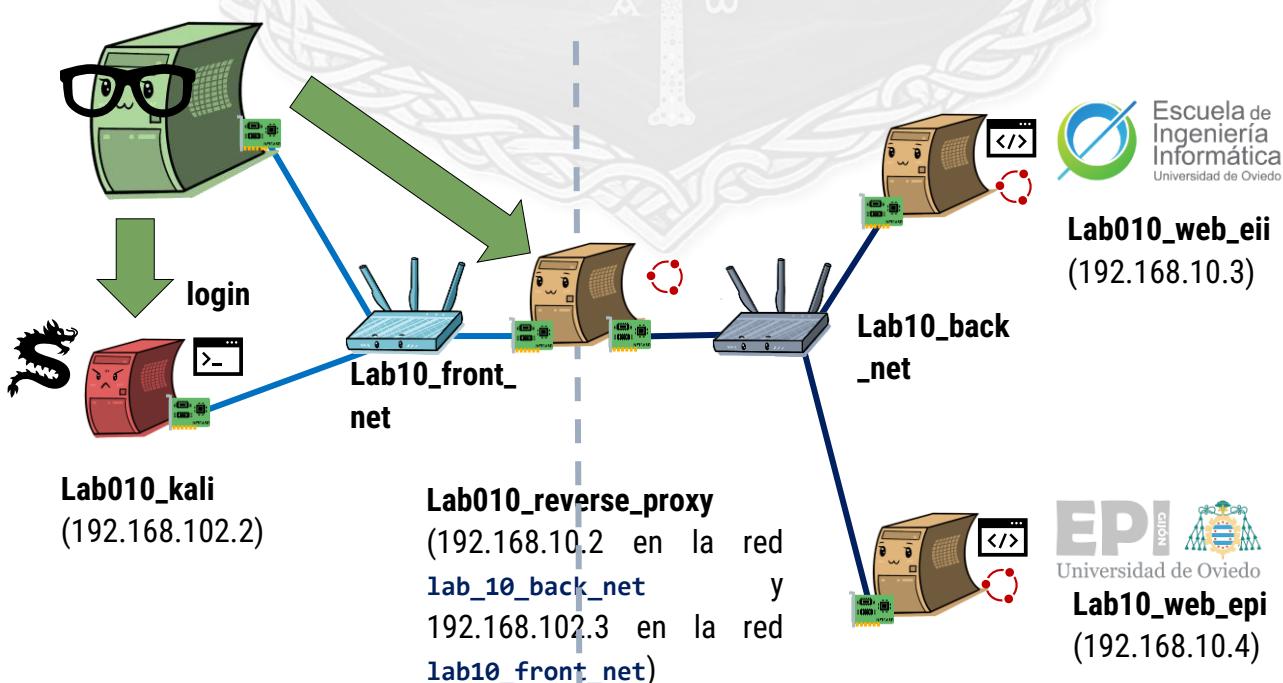
Infraestructura de este Laboratorio

La **infraestructura del Lab 10** proporcionada sigue las mismas convenciones de laboratorios anteriores y tiene los siguientes contenedores cuyas IP se muestran en el diagrama:

- Un contenedor **Kali** con herramientas para "atacar" el proxy. Solo se comunica con el proxy.
- Comunicado con él, un **proxy inverso Apache 2** actuará como intermediario entre el *Kali* y los otros servidores web. Al principio del laboratorio el proxy aún no está configurado, por lo que la comunicación entre *Kali* y los servidores web *backend* no es posible (la habilitaremos en este laboratorio). La máquina proxy tiene IPs en dos redes diferentes, una para comunicarse con el *Kali* y otra para comunicarse con los servidores web. Además de la carpeta para compartir recursos típica, el contenido de `/volume_data/rules` se asigna a la carpeta `/rules`.
- Al otro lado del proxy, comunicados con su propia red privada, tenemos un par de **servidores web**, `lab10_web_eii` y `lab10_web_edi`.
- Además, la máquina virtual *Ubuntu* puede comunicarse con cualquier máquina de la infraestructura, ya que automáticamente tiene conexión con cualquier interfaz de red que hayamos creado.
- **NOTA:** Para evitar problemas de permisos durante los ejercicios, se recomienda trabajar como `root` (`sudo su -`) tanto en el contenedor *Kali* como en el proxy.

Para ejecutar esta infraestructura, descarga el archivo `lab_10.zip` del campus virtual y copia su contenido en la carpeta `lab_sessions` de la infraestructura descomprimida en el laboratorio anterior (deberías tener entonces dos subcarpetas en esta, `lab_08` y `lab_10`). Una vez hecho esto, construye el laboratorio ejecutando primero el script de `prepare_labXX.sh` correspondiente y luego ejecútalo usando `build_labXX.sh`, como en el laboratorio anterior (y como se indica en el documento `lab0B`).

NOTA: Hemos optado por hacer que el build no bloquee y mantener los contenedores vivos incluso después de un reinicio: ya no deberías necesitar hacer build del laboratorio una vez hecho por primera vez, simplemente llama a los scripts de `enter_xxxx.sh` correspondientes y todo debería funcionar bien





Bloque 1: Instalación de un proxy inverso

❖ Ejercicio L10B1_REVPROXY: Apache 2 instalado como un proxy inverso

✿ Descripción de la actividad / Aplicación práctica

Necesitas saber cómo aislar un **servidor web** de accesos desde el exterior de forma que haya un servicio accesible sin exponer su servidor real

✿ Resultados Esperados

Esta actividad finalizará cuando puedas conectarte correctamente a cada servidor web **desde las direcciones URL especificadas de la máquina proxy**.

BACK

1

2

3

4

□ Otra información necesaria para su realización

Uno de los principios fundamentales para crear una infraestructura de máquinas con éxito es la **especialización**: cada máquina tiene un propósito, y todas ellas **deben estar especializadas y optimizadas para cumplirlo**.

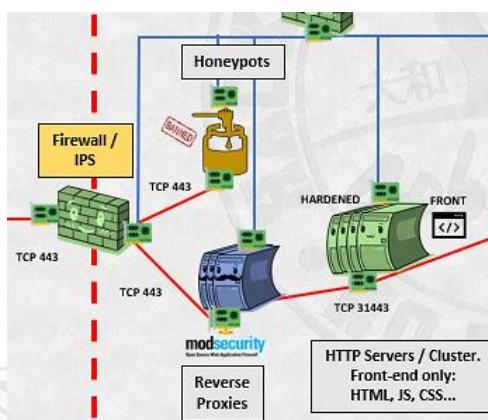
Como piezas de un rompecabezas, estos servidores se juntan en una infraestructura para colaborar entre ellos, por lo que la funcionalidad conjunta de todos ellos logrará un resultado más completo con un mayor nivel de seguridad.

Uno de los principales componentes de una infraestructura segura como la que vimos en la teoría son los **proxies inversos (reverse proxies)**. Estos servidores son una "barrera" entre los clientes y los servicios prestados por la infraestructura:

- Los clientes realmente no pueden saber ningún detalle sobre la infraestructura que les está sirviendo (cuántos servidores hay, software instalado...).
- Los clientes solo pueden acceder a **aquellos servicios que se han creado para ser públicos**. Otros servicios como los internos, auxiliares o de depuración no pueden ser localizados al otro lado del proxy por ningún cliente. Los *proxies inversos* actúan como "**puertas**" (o puntos de acceso únicos) a partes públicas de una infraestructura potencialmente mucho más compleja.

Estos puntos de acceso únicos también suelen estar muy fortificados, por lo que pueden detener ataques para todos los servidores colocados "detrás" de ellos. **Esto significa que tenemos un único punto de defensa para múltiples servidores defendidos, ahorrando tiempo y haciendo especialización de recursos.** De esta manera, las máquinas defendidas solo deben preocuparse por servir contenido (manteniendo, por supuesto, un nivel de seguridad mínimo).

La infraestructura de este laboratorio está hecha para reflejar la presentada en los temas de teoría. Situar un proxy inverso como este “al frente” es una necesidad para evitar problemas provocados por vulnerabilidades potenciales de los servidores web. Concretamente, estamos modelando esta parte.



El contenedor proxy en la [infraestructura del Lab 10](#) ya tiene instalado el servidor web Apache 2 (`sudo apt install apache2`), su módulo para *proxies* (`sudo a2enmod proxy`) y su módulo para *proxies HTTP* (`sudo a2enmod proxy_http`). Por tanto, **en lo relativo a instalación no hay que hacer nada**.

Para configurar Apache2 para que sirva como proxy inverso, debes editar el archivo `/etc/apache2/sites-enabled/000-default.conf` (el archivo de configuración del sitio web predeterminado) para que cada solicitud realizada “desde el frente” a este servidor proxy se redirija a cada servidor web “detrás” dependiendo de la URL especificada.

Por ejemplo, puedes redirigir las solicitudes a “`<IP del proxy>/epi`” y “`<IP del proxy>/eii`” a las IPs indicadas en la figura de ambos servidores web. Para garantizar un proxy inverso **transparente** de las máquinas necesitamos utilizar las directivas `ProxyPass` y `ProxyPassReverse` (https://httpd.apache.org/docs/2.4/mod/mod_proxy.html) sobre las IPs de la red interna. **Ambos deben apuntar a la misma IP del servidor correspondiente**. Por lo tanto, debemos crear dos entradas `Location` en ese archivo con esta estructura, ambas dentro del bloque `VirtualHost` existente:

```
<Location “/<URL>” # Por ejemplo: “/eii”
    ProxyPass “http://<Server IP>”
    ProxyPassReverse “http://<Server IP>”
</Location>
```

Guarda y cierra el archivo y reinicia Apache2 (`service apache2 restart`). Ahora debemos verificar si funciona haciendo peticiones a cualquiera de las URL vistas desde sistema **Kali utilizando la IP del proxy inverso**. También puedes ejecutar *Firefox* desde tu VM *Ubuntu*, ya que tiene comunicación con todas las redes que creamos para este laboratorio, para probar que la navegación va correctamente a las páginas web correspondientes, o simplemente usando `curl` desde el contenedor *Kali*. Si tienes curiosidad, puedes encontrar **configuraciones de proxies más complejas** [aquí](https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html):

NOTA: Por favor, no olvides copiar el archivo `/etc/apache2/sites-enabled/000-default.conf` a `/shared` para conservarlo y restaurar los cambios en caso de que el contenedor proxy se cierre accidentalmente.



Bloque 2. Web Application Firewalls (WAFs)

🛠 Ejercicio L10B2_WAF: Instalación de un WAF

💻 Descripción de la actividad / Aplicación práctica

Necesitas **proteger tus aplicaciones web contra varios tipos de amenazas** sin cambiar el código fuente de la aplicación

🌟 Resultados Esperados

Esta actividad finalizará cuando puedas comprobar con éxito lo siguiente:

- Que **ModSecurity se está ejecutando**, utilizando una regla personalizada y provocando que se active. También puedes encontrar las peticiones bloqueadas en los archivos de log de Apache 2 apropiados (`/var/log/apache2`)
- Que **el motor de reglas de ModSecurity se está usando**, utilizando un ataque falso tipo RCE que active intencionadamente el motor de reglas con las reglas CRS. También puede encontrar las peticiones bloqueadas en los archivos de log de Apache 2 apropiados (`/var/log/apache2`). ¿Este log identifica el tipo de ataque que se ha intentado?
- Que **ModSecurity registra y detiene los intentos de escaneo de Nikto**. Sabiendo que Nikto es bastante rápido, ¿crees que el WAF está obstaculizando los intentos de escaneo y los resultados se obtuvieron de forma mucho más lenta?
- Puedas **lanzar algunos scripts NSE de NMap que funcionan con servidores web** contra el proxy y ver sus resultados. Sabiendo que NMap es rápido en este escenario, ¿crees que el WAF está obstaculizando los intentos de escaneo, por lo que los resultados se obtuvieron de forma mucho más lenta?

📘 Otra información necesaria para su realización

El proxy inverso Apache2 en la **infraestructura del lab 10** también tiene instalado el **Web Application Firewall mod_security2** para interceptar las solicitudes entrantes dirigidas a nuestros sistemas protegidos (se instala con `sudo apt install libapache2-mod-security2`). El WAF está instalado, pero no puede funcionar porque **no tiene reglas adecuadas** que le indiquen cómo examinar las solicitudes y determinar si son peligrosas o no. Estas reglas deben estar instaladas, y esto es precisamente lo que vamos a hacer ahora.

Instalar un OWASP ModSecurity Core Rule Set (CRS) actualizado

Para instalar el ultimo OWASP ModSecurity Core Rule Set primero debemos mover y cambiar el nombre del siguiente archivo de **ModSecurity** para tener una configuración recomendada inicial (`sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf`). Luego, **desde la MV de Ubuntu** descargamos la última versión de **OWASP ModSecurity CRS** desde GitHub: `git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git` (instala `git` si es necesario, pero las máquinas virtuales Vagrant y preconstruidas ya lo tienen). Copiar la carpeta descargada a **Seguridad de Sistemas Informáticos. Proyecto "S-81 'Isaac Peral"**



`/volume_data/rules` de la infraestructura del `lab10` hará que dicha carpeta aparezca en el sistema de archivos del contenedor proxy (`/rules`). Solo entonces podemos continuar.

Una vez dentro del proxy, vete al directorio donde están las reglas descargadas y copia y cambia el nombre `crs-setup.conf.example` a `crs-setup.conf`. Luego copia el directorio `rules/` también junto con su contenido.

```
cd owasp-modsecurity-crs
cp crs-setup.conf.example /etc/modsecurity/crs-setup.conf
cp -r rules/ /etc/modsecurity/
```

Ahora debemos modificar el fichero `/etc/apache2/mods-available/security2.conf` para que coincida con la ruta de los archivos descargados. Esto significa modificar el valor del primer `IncludeOptional` para que coincida con la ruta del archivo `crs-setup.conf` (si no tiene ya un valor correcto) y añadir otra directiva `Include` que apunte al conjunto de reglas. El archivo de configuración debe quedar así (elimina otros contenidos que pueda tener). Reinicia Apache2 de nuevo para que los cambios surtan efecto.

```
<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 /etc/modsecurity/rules/*.conf
</IfModule>
```

NOTA: Una vez más, no olvides copiar este archivo para conservarlo y restaurarlo en caso de errores.

Comprueba que el WAF está en funcionamiento

OWASP CRS es un complemento de *ModSecurity* y las reglas existentes se puedan ampliar. **Este es un tema muy complejo que excede los objetivos de esta asignatura**, pero haremos una demostración para que se vea cómo se puede hacer. Se puede encontrar información detallada sobre la extensión de reglas en el **Wiki oficial de ModSecurity**: [https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-\(v2.x\)](https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x))

Una de las primeras cosas que debemos tener en cuenta es que *ModSecurity* es un software muy complejo, y en ocasiones **sus reglas interfieren o dan falsos positivos sobre solicitudes legítimas** en nuestro entorno de producción. Para comprobar si *ModSecurity* en el proxy está dando protección a los servidores en el *back-end*, podemos realizar las siguientes pruebas:

- Vete hasta la configuración predeterminada de *Apache2* y agrega dos directivas adicionales en el archivo de configuración del sitio web predeterminado (`/etc/apache2/sites-enabled/000-default.conf`).
 - La primera directiva habilita **ModSecurity** en modo de **intercepción** (`On`) detecta amenazas entrantes y **las bloquea**) en lugar del modo de **detección** (`DetectionOnly`) detecta amenazas entrantes pero **solo las registra en un log**, lo cual es útil cuando se prueba que el WAF no está bloqueando solicitudes legítimas).



- La segunda directiva agrega una nueva regla de prueba a **ModSecurity** que se incorporará al conjunto de reglas CRS. Esta regla es muy simple, y se activa cuando alguien usa el parámetro **testarg** en una URL con un valor que contiene la cadena **ssi**. La respuesta del WAF es denegar (**deny**) la solicitud que sirve una página de error con el estado HTTP 403 (**status:403**), registrando el incidente con el mensaje "**regla de prueba SSI disparada!**". Este es solo un ejemplo simple de cómo se crean las reglas WAF, que también verifica que el motor de reglas funciona correctamente.

```
<VirtualHost *:80>
    <Location ...>
        ... # Proxy configuration
    </Location>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    ...
    SecRuleEngine On
    SecRule ARGS:testarg "@contains ssi" "id:1234,deny,status:403,msg:'regla de prueba
SSI disparada!''"
</VirtualHost>
```

BACK

1

2

3

Reinicia Apache 2 de nuevo y, a continuación, accede a la página. Si se carga correctamente, **activa intencionadamente la regla para terminar esta parte del ejercicio**.

- La segunda prueba que vamos a hacer es **comprobar que las reglas CRS se están leyendo**. Para ello, activamos a posta una advertencia de ataque de **ejecución remota de comandos** (RCE), creando una petición que coincide con las reglas CRS que evitan este tipo de ataques, como pasar un parámetro cualquiera con valor **/bin/bash**

WAF contra "el mal"

Usando el siguiente *cheatsheet*, ejecuta el escáner de servidores web *Nikto* instalado en el contenedor de ataque *Kali* de la [infraestructura del Lab 10](#) contra el proxy para ver qué sucede.

Otra prueba que podemos hacer es usar algunos de los scripts HTTP **nmap** NSE que probamos en el laboratorio anterior contra el proxy, y ver si el resultado cambia o los intentos son registrados por **ModSecurity**.



Nikto 2.1.6 Cheatsheet (ingenieriainformatica.uniovi.es)					
Lightweight web server vulnerability scanner					
https://cirt.net/Nikto2					
GENERAL USAGE					EVASION OPTIONS
nikto -host <url> [options]					1 Random URI encoding (non-UTF8)¶
					2 Directory self-reference (./)¶
NOTES					3 Premature URL ending¶
+ means that the option requires a value					4 Prepend Long random string¶
Options in stronger bold font are detailed in separate tables					5 Fake parameter¶
OPTIONS					6 TAB as request spacer¶
-config+: Use this config file					7 Change the case of the URL¶
-dbcheck+: check database and other key files for syntax errors					8 Use Windows directory separator (\)¶
-Display+: Turn on/off display outputs					A Use a carriage return (0xd) as a request spacer¶
-evasion: Encoding technique to use to avoid WAFs, etc.					B Use binary value 0xb as a request spacer¶
-Format+: save file (-o) format					
-Help: Extended help information					
-host+: target host/URL					
-id+: Host authentication to use, format is id:pass or id:pass:realm					
-list-plugins: List all available plugins					
-mutate: Guess additional file names					
-no404: Disables 404 checks					
-nssl: Disables using SSL					
-output+: Write output to this file					
-Plugins+: List of plugins to run (default: ALL)					
-port+: Port to use (default 80)					
-root+: Prepend root value to all requests, format is /directory					
-ssl: Force ssl mode on port					
-timeout+: Timeout for requests (default 10 seconds)					
-Tuning+: Scan tuning					
-update: Update databases and plugins from CIRT.net					
-Version: Print plugin and database versions					
-vhost+: Virtual host (for Host header)					
EXAMPLES					MUTATE OPTIONS
nikto -Display 1234EP -o report.html -Format htm -Tuning 123bde -host					1 Test all files with all root directories
192.168.20.10					2 Guess for password file names
					3 Enumerate user names via Apache (~user type requests)
					4 Enumerate user names via cgiwrap (/cgi-bin/cgiwrap/~user type requests)
					5 Attempt to brute force sub-domain names, assume that the host name is the parent domain
					6 Attempt to guess directory names from the supplied dictionary file
EXAMPLES					TUNING OPTIONS
nikto -Display 1234EP -o report.html -Format htm -Tuning 123bde -host					1 Interesting File / Seen in Logs
192.168.20.10					2 Misconfiguration / Default File
					3 Information Disclosure
					4 Injection (XSS/Script/HTML)
					5 Remote File Retrieval - Inside Web Root
					6 Denial of Service
					7 Remote File Retrieval - Server Wide
					8 Command Execution / Remote Shell
					9 SQL Injection
					0 File Upload
					a Authentication Bypass
					b Software Identification
					c Remote Source Inclusion
					d WebService
					e Administrative Console
					x Reverse Tuning Options (i.e., include all except specified)

BACK

1

2

3



🛠 Ejercicio L10B2_WAFPROT: WAF contra ataques web

👤 Descripción de la actividad / Aplicación práctica

Necesitas comprobar si el **WAF** está realmente bloqueando peticiones web maliciosas del estilo a las que se mencionan en el último tema de teoría

⌚ Resultados Esperados

Esta actividad finalizará cuando puedas comprobar con éxito que **ModSecurity registra y detiene los intentos de ataque XSS y de inyección SQL**. ¿El log identifica el tipo de ataque que se ha intentado hacer?

📋 Otra información necesaria para su realización

Seguiremos un procedimiento muy similar al anterior para ver si el WAF se activa cuando se realiza un intento de ataque XSS. "Provoca" al WAF creando una solicitud HTTP falsa al proxy que contenga JavaScript y/o SQL válidos (por ejemplo, como el valor de un parámetro que te inventes) para ver si están bloqueados.



Bloque 3: Herramientas AST para el desarrollo de aplicaciones

🛠 Ejercicio L10B3_SASTSCA: Herramientas de detección automatizada de vulnerabilidades en aplicaciones (SAST, SCA)

💻 Descripción de la actividad / Aplicación práctica

Necesitas **detectar automáticamente vulnerabilidades de dependencias y de código fuente** en tus aplicaciones

🏆 Resultados Esperados

Esta actividad finalizará cuando puedas hacer estas dos operaciones además de responder las preguntas que se plantean a lo largo del ejercicio:

- **Hacer con éxito un análisis SAST** de una aplicación con *SonarQube* o *GitHub* para buscar problemas y vulnerabilidades de seguridad e interpretar correctamente los hallazgos.
- Analizar con éxito una aplicación con *SonarQube* o *GitHub* para **localizar dependencias vulnerables** y analizar el informe generado con información sobre ellas.

📋 Otra información necesaria para su realización

SAST automatizado con *SonarQube*

SonarQube (<https://www.sonarqube.org/>) es una **herramienta automatizada para la revisión de la calidad del código y su seguridad** que realiza **análisis de código estático (SAST)**, como se explica en los temas de teoría del código fuente de nuestra aplicación.

Admite alrededor de 20 lenguajes de programación diferentes (principalmente Java o C#) y busca bugs, vulnerabilidades, security hotspots, code smells, métricas de calidad de código, etc.

Se integra con *Maven*, *Gradle*, y otras plataformas de construcción de aplicaciones también utilizadas en otros cursos como **ASW**, **ASR** o **SDI** así que, si tus proyectos las usan, *SonarQube* será muy fácil de configurar y usar. También acepta módulos que le permiten hacer otras funcionalidades.

Los dos ejercicios siguientes no usan la infraestructura del Laboratorio 10, así que mejor detenla para ahorrar recursos.

Esta herramienta también puede identificar errores que no se han detectado en las pruebas. Como eso ya lo hacéis en otro curso, nosotros nos vamos a centrar en las **vulnerabilidades** que puedan estar presentes en el código fuente. De hecho, podéis usar el *SonarCloud* del otro curso (una versión en nube de la herramienta que vamos a usar aquí) para mirar los apartados a los que haremos referencia más abajo en los proyectos que hayáis probado en el mismo, ya que es el resultado que buscamos.



Para usar SonarQube necesitamos seguir estos pasos desde tu máquina virtual **Ubuntu**:

- **Elige un proyecto en un lenguaje soportado** para analizar y cópialo dentro de la MV del curso (preferiblemente en Java ya que debe poder compilarse en la MV, usando *Maven* o *Gradle*). Puede ser cualquier aplicación que tengas de otros cursos (como la que usaste con *SonarCloud*). Si no tienes una, puedes usar este <https://github.com/srebreni3/simple-maven-java-app> o este <https://github.com/javaparser/javaparser-maven-sample> para revisar el sistema, pero son muy simples y no contienen vulnerabilidades.

En esta práctica buscamos alguna que compile en la MV y que tenga un fichero pom.xml de Maven, básicamente

- Instalar **Maven en la máquina virtual de Ubuntu** (`sudo apt install maven`)

Ejecute un **contenedor SonarQube** siguiendo la sección "**Enlace** de puertos" de esta URL: https://hub.docker.com/_/sonarqube, algo como `docker run --name sonarqube-custom -p 9000:9000 sonarqube:community`. La primera vez que lo ejecuta tarda un tiempo en completarse.

Por favor, no olvides abrir el puerto usado por la herramienta si tienes un firewall habilitado, o deshabilite el firewall temporalmente para realizar estas pruebas

- Inicia sesión en <http://localhost:9000> (o el puerto que hayas elegido) con credenciales de administrador del sistema (login=**admin**, password=**admin**). Cambiar la contraseña en el primer inicio de sesión es obligatorio.
- Para analizar un proyecto:
 - Haz clic en **Create new project manually**.
 - Asigna al proyecto una **Project key** y un **Display name** y haz clic en el **Configure** (ej. **SAST_SSI**).
 - Haga clic en **Analyze your repository locally**.

The screenshot shows the SonarQube interface. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. Below the navigation bar, it shows a project named 'ssi_test' with a 'master' branch. Underneath, there are tabs for Overview, Issues, Security Hotspots, Measures, Code, and Activity. The 'Overview' tab is selected. A section titled 'How do you want to analyze your repository?' asks if the user wants to integrate with CI. It lists options for Jenkins, GitHub Actions, Bitbucket Pipelines, GitLab CI, Azure Pipelines, and Other CI. Below this, a note says 'Are you just testing or have an advanced use-case? Analyze your project locally.' and shows a button labeled 'Locally' with a small icon of a computer monitor. This 'Locally' button is highlighted with a large red rectangular box.

Figura 1. El GUI puede ser un poco distinto por los cambios entre versiones, pero las opciones siguen ahí :)

- Mira el token generado y haz clic en continuar. Luego, elije **Maven**. Esto es porque los proyectos de prueba propuestos se construyen con *Maven* pero, como ves, hay muchas otras



opciones que puedes usar en el futuro (como por ejemplo, tu TFG 😊) o de la aplicación de prueba que hayas elegido.

- **Ahora te mostrará un comando para analizar la aplicación asociada con el proyecto SonarQube que acabas de crear.** Ten en cuenta que el token y la clave del proyecto son los que se generaron al configurar el análisis.
- Vete a la carpeta del proyecto en tu máquina virtual y ejecuta el comando que te mostró la interfaz de SonarQube dentro de ella para analizar el proyecto. El comando será algo como:

```
mvn sonar:sonar \
-Dsonar.projectKey=testPrimeroSSI \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.login=686c129bbb6ac40a02534d7eb02fb8393037482c
```

- Una vez finalizado el análisis, la interfaz de usuario de SonarQube se actualizará automáticamente con los resultados de este. Tendrás categorías de **Bugs** y **Code Smells** con elementos que están más relacionados con otros cursos. Las categorías más vinculadas con nuestra asignatura son **Vulnerabilities** y la pestaña **Security Hotspots**. Mira si dan algún resultado (algo que podría pasar si has usado alguna aplicación tuya, no las de ejemplo que solo valen para probar que todo funciona).

SCA automatizado con SonarQube

SonarQube es una herramienta SAST, y **no realiza funcionalidades SCA por sí misma**. Sin embargo, su sistema de *plugins* nos permite incorporar una herramienta SCA completa y de buena reputación en el *pipeline* de construcción. Esta herramienta es el software **OWASP Dependency Check**: <https://github.com/dependency-check/DependencyCheck>. Para ello, sigue estos pasos:

- Si has usado un proyecto *Maven* para probar, localiza su archivo **pom.xml** y asegúrate de añadir las líneas que aparecen en rojo aquí. Estas son las instrucciones oficiales del repositorio de *GitHub*. Si usaste algo diferente a *Maven*, esa web también tiene instrucciones para otros plugins:

```
...
<plugin>
    ...
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.3.0.M2</version>
    <configuration>
        <stopKey>CTRL+C</stopKey>
        <stopPort>8999</stopPort>
        <systemProperties>
            <systemProperty>
                <name>xwork.loggerFactory</name>
                <value>com.opensymphony.xwork2.util.logging.log4j2.Log4j2Logger
Factory</value>
            </systemProperty>
        </systemProperties>
        <scanIntervalSeconds>10</scanIntervalSeconds>
        <webAppSourceDirectory>${basedir}/src/main/webapp/</webAppSourceDirecto
ry>
        <webAppConfig>
            <descriptor>${basedir}/src/main/webapp/WEB-INF/web.xml</descriptor>
        </webAppConfig>
    </configuration>
</plugin>
<plugin>
```



```
<groupId>org.owasp</groupId>
<artifactId>dependency-check-maven</artifactId>
<executions>
    <execution>
        <goals>
            <goal>check</goal>
        </goals>
    </execution>
</executions>
</plugin>
</plugins>
...
```

- Repite el mismo proceso del análisis SAST anterior para iniciar el análisis SonarQube. Ahora, como parte del análisis automatizado de SonarQube, también se hará un análisis SCA de las dependencias del proyecto de prueba. Esto requiere una cantidad sustancial de tiempo (necesita descargar toda la información de CVEs disponible) y también puede bloquearse debido a la falta de recursos, especialmente si no pudiste aumentar la RAM de la máquina virtual. En cualquier caso, se generará un informe en: `<la carpeta en la que descomprimiste la aplicación de prueba>/dvja-master/target/dependency-check-report.html`

BACK

1
2
3

SAST y SCA con GitHub

Para hacer el mismo tipo de escaneos con *GitHub* necesitas seguir unos pasos sencillos:

- Asegúrate de tener **una cuenta de GitHub** (deberías tener una de otros cursos, pero si no, créala).
- Puedes escanear **cualquier proyecto de software que tengas en GitHub**. Si no tienes uno, puedes **hacer fork** de cualquiera en tu repositorio para escanearlo, como <https://github.com/appsecco/dvja>
- Vete a la pestaña **Security** de tu proyecto y habilita cualquier opción relacionada con la seguridad. **El Dependabot** es el SCA, el **Code scanning** es el SAST. Hay más funciones (evaluación de vulnerabilidades asistida por IA, escaneo de secretos...). Habilítalo todo, es gratis!. El resultado debería ser algo como:

The screenshot shows the GitHub Security overview page for a repository. At the top, there are navigation links: 'Issues' (highlighted), 'Wiki', 'Security' (which is active), '31', 'Insights', and 'Settings'. Below this, a yellow banner says 'Make sure you still have them stored somewhere safe by viewing and downloading them again.' with a 'View' button. The main section is titled 'Security overview'.

Setting	Status	Action
Security policy	Disabled	Set up a security policy
Security advisories	Enabled	View security advisories
Private vulnerability reporting	Disabled	Enable vulnerability reporting
Dependabot alerts	Enabled	View Dependabot alerts
Code scanning alerts	Enabled	View alerts
Secret scanning alerts	Enabled	View detected secrets

- Guarda la configuración y **espera a que finalicen las GitHub Actions** que controlan estas comprobaciones de seguridad. En mi caso encontró 28 vulnerabilidades debido a que este proyecto está usando dependencias desactualizadas y 3 problemas de seguridad en mi código. Haga clic en cada categoría para verlas



Code Pull requests 1 Actions Projects Wiki Security 31 Insights Settings

Your recovery codes have not been saved in the past year. Make sure you still have them stored somewhere safe by viewing and downloading them again.

[View](#)

[Overview](#)

Security overview

- Reporting**
 - Policy**
 - Advisories**
- Vulnerability alerts**
 - Dependabot** 28
 - Code scanning** 3
 - Secret scanning**

Setting	Status	Action
Security policy	Disabled	Set up a security policy
Security advisories	Enabled	View security advisories
Private vulnerability reporting	Disabled	Enable vulnerability reporting
Dependabot alerts	Enabled	View Dependabot alerts
Code scanning alerts	Enabled	View alerts
Secret scanning alerts	Enabled	View detected secrets

- Obviamente, los resultados pueden variar dependiendo del proyecto que escanees.

[BACK](#)

Dependabot alerts
Dependency files checked yesterday

Auto-triage your alerts
Control how Dependabot opens pull requests, ignores false positives and snoozes alerts. Rules can be enforced at the organization level. Free for open source and available for private repos through [GitHub Advanced Security](#). [Learn more about auto-triage](#)

is:open

28 Open 0 Closed

Package	Ecosystem	Manifest	Severity	Sort
Apache Struts file upload logic is flawed	Critical	#28 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Remote code injection in Log4j	Critical	#24 opened yesterday • Detected in org.apache.logging.log4j.log4j-core (Maven) • pom.xml	3	
Apache Struts vulnerable to path traversal	Critical	#22 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Expression Language Injection in Apache Struts	Critical	#11 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Incomplete fix for Apache Log4j vulnerability	Critical	#9 opened yesterday • Detected in org.apache.logging.log4j.log4j-core (Maven) • pom.xml	3	
Improperly Controlled Modification of Dynamically-Determined Object Attributes in Apache Struts	Critical	#7 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Apache Struts vulnerable to remote arbitrary command execution due to improper input validation	Critical	#6 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Remote code execution in Apache Struts	Critical	#5 opened yesterday • Detected in org.apache.struts.struts2-core (Maven) • pom.xml	4	
Deserialization of Untrusted Data in Log4j	Critical	#4 opened yesterday • Detected in org.apache.logging.log4j.log4j-core (Maven) • pom.xml	3	

🛠 Ejercicio L10B3_HARDPROXY: Combinación del proxy con técnicas anteriores

💻 Descripción de la actividad / Aplicación práctica

Entender cómo una máquina proxy puede ser un punto estratégico donde puedes incrementar las medidas de seguridad de tu infraestructura por encima de otras partes de esta.



💡 Resultados Esperados

Esta actividad finalizará cuando **sepas qué hacer para proporcionar una mayor seguridad a la máquina proxy** utilizando técnicas que vimos en laboratorios anteriores.

📋 Otra información necesaria para su realización

Una vez que tengamos este proxy con seguridad mejorada creado, y aprovechando las actividades que realizamos en laboratorios anteriores, piensa en cómo las combinarías para que la protección pueda ser aún mayor en un escenario real. Para hacer eso, responde estas preguntas:

- *¿Qué harías con el sistema operativo de una máquina que haga de proxy?*
- Si el proxy debe tener instalada una conexión remota **ssh** pública por razones de administración, *¿qué medidas de seguridad adicionales habilitarías para este servicio?*
- Ahora que tenemos un proxy web expuesto, *¿instalarías fail2ban en él?* Si la respuesta es sí, *¿qué pasos adicionales tomarías de los que vimos en un laboratorio anterior?*
- *¿Permitirías escaneos nmap en esta máquina? ¿Qué harías para evitarlos?*
- Si no quieres una conexión **ssh** pública en el proxy, pero aun así quieras poder conectarse a él a través de **ssh** para fines de administración, *¿qué harías?*
- Si te preocupan las vulnerabilidades en el código de tus aplicaciones, *¿qué contramedidas utilizarías para prevenirlas tanto como sea posible?*
- Si puedes responder a las preguntas anteriores, *¿qué opinas sobre la seguridad de la máquina resultante y la protección que proporciona a la infraestructura subyacente?*

◀
BACK

1

2

3

📝

🛠 Ejercicio L10B3_DAST: Uso de un DAST contra una aplicación

💻 Descripción de la actividad / Aplicación práctica

Esta actividad te enseña a **hacer pruebas DAST sobre una aplicación web**.

💡 Resultados Esperados

Esta actividad se considera terminada **cuando seas capaz de usar ZAP para hacer un escaneo DAST de una web de test**, tanto de forma activa como manual, usando además el modo agresivo para ver si se detectan problemas. Eres además capaz de entender el tipo de errores que puede descubrir frente a soluciones SAST o SCA e interpretar los informes generados. Puedes responder a estas preguntas:

- *¿Consideras interesante introducir herramientas como esta en un pipeline CI/CD de construcción de tu aplicación?*
- *¿Crees que es interesante presentar un informe de ZAP hecho sobre la versión final de tu producto o TFG a tus inversores / clientes / tribunal?*

📋 Otra información necesaria para su realización

El uso de ZAP como DAST requiere una serie de pasos que se van a explicar a continuación.



Introducción

OWASP ZAP (<https://www.zaproxy.org/getting-started/>) es una herramienta gratuita típicamente usada para hacer *pentesting* de aplicaciones web. No obstante, **también es una herramienta DAST** según cómo se use.

Recuerda que, tal y como vimos en la teoría, las DAST son herramientas de caja negra que “atacan” aplicaciones desde fuera de las mismas, introduciendo posibles errores para detectar vulnerabilidades. ZAP en sí es un proxy que se sitúa entre medias del navegador y la aplicación examinada, interceptando peticiones entre ambos, permitiendo su modificación, y enviándolas posteriormente a la web.

No obstante, ZAP nos dio muchos problemas al instalarlo *Ubuntu*, ya que se bloqueaba aleatoriamente durante la ejecución. Por tanto, para evitarlos, la mejor solución es **descargar una MV VirtualBox preempaquetada oficial de Kali Linux** desde aquí: <https://www.kali.org/get-kali/#kali-virtual-machines>, y seguir estas instrucciones:

- **Descomprimirla** (usa 7-Zip)
- **Haz doble clic en el archivo .vdi de la MV** (el que tiene un ícono de cubo azul) para abrir la MV en *VirtualBox*. Aumentar la memoria RAM a 4 Gb ayudará a que todo funcione sin problemas.
- **Inicia sesión** usando **kali/kali** como nombre de usuario y contraseña
- (para poner el teclado en español: setxkbmap -layout es)
- **Vete a una terminal** y ejecuta **sudo apt-get update** para actualizar todas las direcciones de los paquetes. Actualizar todo puede llevar mucho tiempo, y no es necesario para este laboratorio
- **Instala ZAP** con **sudo apt install zaproxy**
- **Ábrelo** usando el ícono del programa en el menú de software de *Kali*

Lo cierto es que para poder probar este DAST **necesitaremos pues una aplicación “víctima”**, y podemos usar cualquiera de otra asignatura para ello que tengamos a mano y de la que **seamos propietarios o hayamos logrado una autorización por escrito (no hagas esto a una aplicación web pública cualquiera)**. Si no fuera el caso, una buena forma de probarla es instalando y configurando la **Damn Vulnerable Web App (DVWA)** como se indica aquí: <https://hub.docker.com/r/vulnerables/web-dvwa/>. Necesitaremos *Docker* para ello: **docker run --rm -it -p 8081:80 vulnerables/web-dvwa** (si usas un *firewall*, abre el puerto **8081** y el **8080**, que es el que usa ZAP).

Si Kali no tiene docker instalado, puede instalarlo con sudo apt install docker.io

Escaneo activo

La forma más directa de emplear esta herramienta DAST es mediante su opción *Quick Start-Automated Scan*. Le damos simplemente la URL de la web a analizar y tras un tiempo **nos dará una lista con los problemas de seguridad que ha encontrado en la misma**, que podremos analizar clicando en cada uno de ellos para saber sus detalles, gravedad, etc. **Veremos también que ha hecho spidering de la aplicación**, es decir, que nos mostrará un árbol de páginas que componen el sitio examinado.



The screenshot shows the 'Automated Scan' tab in ZAP. It has a URL input field containing 'http://localhost:8080/' and several configuration options: 'Use traditional spider:' checked, 'Use ajax spider:' with a dropdown set to 'Firefox Headless', and an 'Attack' button. A progress message at the bottom says 'Attack complete - see the Alerts tab for details of any issues found'.

Si todo ha ido correctamente, reconoceremos algunos de los problemas que hemos descrito en la teoría, a los que tendríamos que buscar solución antes de poner la aplicación en producción.

Escaneo manual

Si bien el escaneo activo es una forma muy sencilla y directa de usar la herramienta DAST, tiene un problema del que probablemente ya te habrás dado cuenta: **si tienes una pantalla de login para una zona privada, la herramienta no podrá pasar a examinar esa zona privada porque no sabe las credenciales**. Para resolver esto, una de las opciones más sencillas es **hacer un escaneo manual**: Nosotros navegamos por la aplicación en un navegador que ZAP nos muestra, nos autenticamos normalmente (estamos en un **escenario de test de seguridad**, se supone que tenemos credenciales) y el proxy va a haciendo pruebas de seguridad (y *spidering*) a medida que navegamos por la aplicación normalmente.

Así podemos navegar a todas las partes que nos interesen y acceder a páginas que por alguna razón no pueda llegar el spider de la herramienta.

The screenshot shows the 'Manual Explore' tab in ZAP. It has a URL input field containing 'http://localhost:8080/' and configuration options: 'Enable HUD:' checked, and 'Explore your application:' set to 'Launch Browser Firefox'. A note at the bottom says 'You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.'

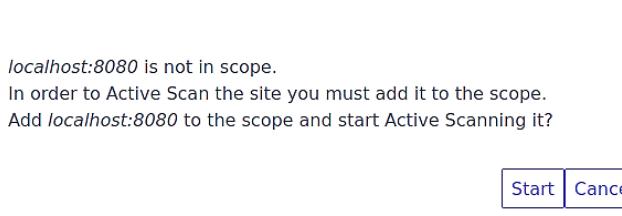
Hay una cosa que debemos tener en cuenta sobre el navegador que nos muestra la herramienta: es el navegador predeterminado del sistema con un plugin llamado **HUD**, que nos saca **en el lado izquierdo las vulnerabilidades de la página web** que estamos visitando en esos momentos (con su gravedad con un típico código de colores en banderas) y **a la derecha las acumuladas por todas las webs visitadas del sitio hasta el momento**.

Como veremos, además ZAP sigue haciendo su trabajo "por debajo", como lo hacía con el escaneo activo



Mientras navegamos por la web ahora vemos controles adicionales que nos indican el estado de los hallazgos de la herramienta en la misma. Para entender bien lo que está pasando mientras navegamos por la web de pruebas, tenemos que considerar lo siguiente:

- Lo que va encontrando se clasifica por gravedad en la botonera superpuesta del lado izquierdo de la web.
- Al lado derecho de la web podemos **activar el escaneo activo anterior en cualquier momento**, señal de que podemos lanzarlo una vez superada la pantalla de *login*
- En función de lo que hayamos hecho, la herramienta dará este aviso, que simplemente aceptamos y pulsamos en “Start” para hacer el escaneo, cuyo progreso veremos sobre los iconos del HUD.



- Las vulnerabilidades del escaneo manual o híbrido aparecerán de nuevo en la interfaz de ZAP.

The screenshot shows the ZAP interface with the 'Alerts' tab selected. A tooltip explains: 'Full details of any selected alert will be displayed here. You can manually add alerts by right clicking on the relevant item in the history or search results. You can also edit existing alerts by double clicking on them.' The list of alerts includes: Absence of Anti-CSRF Tokens (5), CSP: Wildcard Directive (5), Content Security Policy (CSP) Header Not Set, Cross-Domain Misconfiguration (32), Directory Browsing (5), Missing Anti-clickjacking Header (7), Cookie No HttpOnly Flag (2), Cookie without SameSite Attribute (2), Timestamp Disclosure - Unix (155), X-Content-Type-Options Header Missing (1453), Charset Mismatch, Loosely Scoped Cookie, and Do Not examine Cache-control Directives (22).

- **Este modo de trabajo permite usar una opción especial, el “Attack Mode”,** representado por una mirilla de un arma y activado haciendo clic en la misma. El *Attack Mode* funciona a medida que vamos navegando por las páginas y se puede hacer un *active scan* en paralelo. Esto es una forma especial de trabajo de ZAP **donde hará pruebas intrusivas a todas las páginas** que pueda llegar a partir de las que estemos navegando, aumentando por tanto las opciones de encontrar más problemas, pero también el riesgo de “romper” algo en la aplicación por meter datos corruptos, hacer *fuzzing*, etc. Por ese motivo, antes de usar este “*Attack Mode*” **se recomienda encarecidamente hacer dos cosas**:
 1. **Desconectar la red de la máquina de pruebas por si acaso, si es posible.** Es muy frecuente que muchas páginas usen servicios de terceros para algo (por ejemplo, publicidad, *trackers*...) y se podrían llevar un ataque por accidente si no tenemos cuidado. Por supuesto, **solo se podría hacer si la página es capaz de funcionar de esta forma**. En cualquier caso, **nunca navegar a la web de un servicio de terceros en este modo, porque podrías cometer un delito**.
 2. **Hacer las pruebas sobre un clon de la web** para evitar interacciones no deseadas y corrupción de datos por las acciones de ZAP.- Finalmente, no olvides que ZAP tiene la opción de **obtener informes (Reports)** donde se resumen todas las vulnerabilidades encontrar, su gravedad y otra información, ideal para pasar al equipo de desarrollo para que procedan a resolverlas.



Insignias y autoevaluación

NOTA: Tienes una versión de esta tabla de insignias en formato editable disponible en el *Campus Virtual*. Puedes usar este archivo para crear un documento en el formato que deseas y tomar notas extendidas de tus actividades para crear un log de lo que has hecho. Recuerda que el material que elaboras se puede llevar a los exámenes de laboratorio.

Nivel de insignia	Desbloqueado cuando	¿Desbloqueado?
	Entiendes lo que es un proxy inverso. Puede responder a esta pregunta: <i>¿Un proxy inverso realmente sirve sitios web por sí mismo?</i>	
	Entiendes cuál es la funcionalidad principal de un WAF	
	Sabes cómo iniciar escaneos web usando nikto	
	Puedes verificar si <i>ModSecurity</i> también proporciona protección contra algunos escaneos nmap	
	Comprendes el propósito de una herramienta SAST y el tipo de resultados que produce	
	Comprendes el propósito de una herramienta SCA y el tipo de resultados que produce	
	Puedes responder a las preguntas: <i>¿Un proxy inverso aísla completamente una infraestructura de servicios de sus clientes potenciales? ¿Cuál es la razón de usar dos rangos de red totalmente diferentes?</i>	
	Puede responder a estas preguntas: <i>¿La regla personalizada que utilizamos está probando el motor ModSecurity en sí o las reglas CRS que descargamos? ¿Es esto suficiente para probar que toda la infraestructura está funcionando?</i>	
	Puede responder a estas preguntas: <i>¿El falso ataque RCE que hicimos está probando el motor ModSecurity en sí o las reglas CRS que descargamos? ¿Es esto suficiente para probar que toda la infraestructura está funcionando?</i>	
	Puedes responder a estas preguntas: <i>¿Se registran las solicitudes bloqueadas de ModSecurity? En ese caso, ¿en qué fichero?</i>	
	Puede responder a esta pregunta: <i>¿Protege un WAF contra los intentos de ataque automatizados?</i>	
	Puede responder a esta pregunta: <i>¿Protege un WAF contra los intentos típicos de ataque web (XSS, inyección SQL...)? ¿Identifica el tipo de intento de ataque? Por lo tanto, ¿puede una GUI potencial que lea los logs del WAF mostrar a sus usuarios los tipos de ataque que se están produciendo?</i>	
	Comprendes cómo se pueden combinar un NIDS y un forward proxy para detectar máquinas comprometidas en una red interna	



	Puedes analizar el código fuente de una aplicación utilizando SonarQube. Puedes responder a estas preguntas: <i>¿Crees que este tipo de herramientas localizan todos los errores posibles? Si no es así, ¿crees que conviene usarlas?</i>	
	Puedes configurar correctamente un WAF ModSecurity2 que funcione protegiendo una infraestructura de servidor	
	Teniendo en cuenta que un WAF identifica un ataque solo si la sintaxis del lenguaje en el que se codifica el mismo es correcta (XSS, SQL Injection...), razona sobre cuál podría ser la técnica principal que se puede utilizar para eludir la protección de un WAF	
	Puedes responder a estas preguntas: <i>¿Por qué es importante un proxy inverso desde el punto de vista de la seguridad? ¿Crees que este concepto es como la encapsulación en la programación orientada a objetos?</i>	
	Puedes responder a esta pregunta: <i>¿Crees que los WAF proporcionan la máxima protección contra ataques, o deberían tratarse como una capa adicional de protección?</i>	
	Sabes cómo combinar técnicas que vimos en laboratorios anteriores para mejorar aún más la seguridad de un proxy en más escenarios.	
	Sabes cómo usar una herramienta DAST como ZAP para comprobar las vulnerabilidades de una aplicación web en ejecución y examinar sus páginas en busca de vulnerabilidades de manera automatizada o manual	
	Comprendes que una aplicación entregada con un informe SCA, SAST y DAST correctos siempre ofrece más garantías que la carencia de los resultados de herramientas de esta clase	
	Entiendes claramente el tipo de vulnerabilidades que pueden ser prevenidas por un WAF como ModSecurity frente a las preventidas por una herramienta de análisis estático como SonarQube	

← BACK
 1
 2
 3
 ✎