Fuente: Bing Chat AI

LABORATORIO 10. DEFENDIENDO APLICACIONES WEB

DEPARTAMENTO DE INFORMÁTICA. UNIVERSIDAD DE OVIEDO

Seguridad de Sistemas Informáticos | 2023 – 2024 (V4.0 *"S-81 Isaac Peral"*)

Contenido

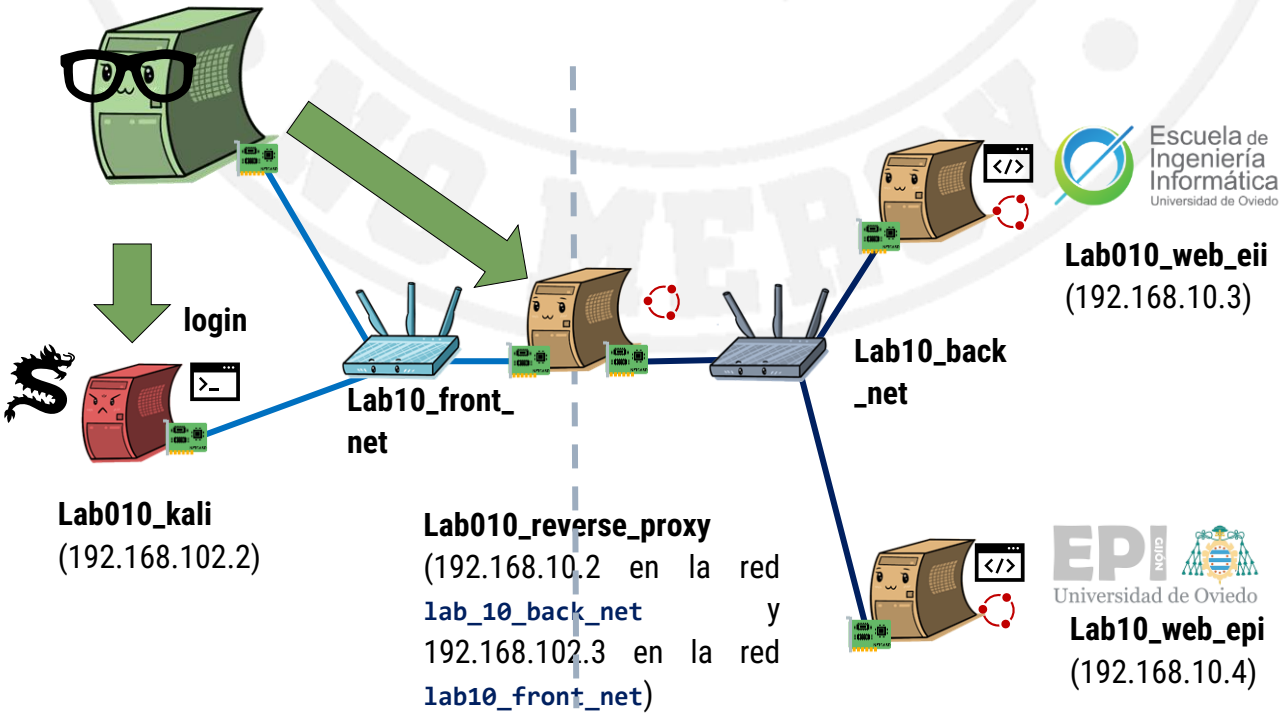
Infraestructura de este Laboratorio

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 de ataque" **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 de 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_epi`.
- 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.

NOTA: Si tuvieras problemas de espacio o de construcción de los laboratorios, hemos creado dos scripts que borran todas las imágenes de *Docker* ya creadas. Si un `prepare__labXX.sh` te da problemas, por favor borra todas las imágenes y ejecuta de nuevo `prepare__labXX.sh` para obligar al sistema a reconstruirlo todo, lo que debería solucionar todos los problemas. Estos scripts están en el *Campus Virtual*. Una vez descargados a la MV, para ejecutarlos deberías hacer:

- `sh borrar_imagenes_ubuntu.sh`: Para borrar las imágenes *Ubuntu*
- `sh borrar_imagenes_kali.sh`: Para borrar las imágenes *Kali*



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**.

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**.

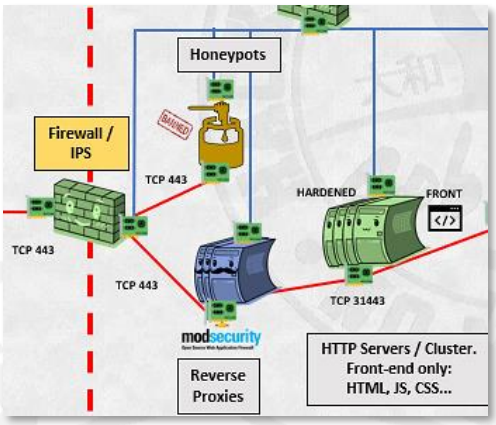
Como piezas de un rompecabezas, estos servidores se juntan en una infraestructura para colaborar entre ellos, por lo que la funcionalidad conjunta de todos ellos logrará un resultado más completo con un mayor nivel de seguridad.

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 cerrase 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 predeterminada 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

`/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.

> 🔍 ***NOTA:** También os proporcionamos un conjunto de reglas de muestra (desactualizado, pero suficiente para nuestros propósitos) en un archivo zip como parte del contenido de la infraestructura de Lab 10*

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)

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.

> 🔍 ***Por ejemplo, dependiendo de las reglas que se hayan activado, ModSecurity podría prohibir el acceso a máquinas cuando para ello se usan sus IP directamente, ya que esto se considera el comienzo de un intento de ataque (en situaciones reales esto lo hacen principalmente escáneres, arañas o bots). En una situación real, esto se puede evitar utilizando un DNS o modificando el archivo `/etc/hosts` para dar un nombre que se resolverá al acceder a la IP. No deberías encontrar este problema en este laboratorio.***

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>
```

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 coincida 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.

Nikto 2.1.6 Cheatsheet (ingenieriainformatica.uniovi.es)
Lightweight web server vulnerability scanner

https://cirt.net/Nikto2

GENERAL USAGE
nikto -host <url> [options]

NOTES
+ means that the option requires a value
Options in stronger bold font are detailed in separate tables

OPTIONS
-config+: Use this config file
-dbcheck: check database and other key files for syntax errors
-Display+: Turn on/off display outputs
-evasion: Encoding technique to use to avoid WAFs, etc.
-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
-nossl: 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
nikto -Display 1234EP -o report.html -Format htm -Tuning 123bde -host 192.168.20.10

EVASION OPTIONS	
1	Random URI encoding (non-UTF8)
2	Directory self-reference (/./)
3	Premature URL ending
4	Prepend long random string
5	Fake parameter
6	TAB as request spacer
7	Change the case of the URL
8	Use Windows directory separator (\)
A	Use a carriage return (0x0d) as a request spacer
B	Use binary value 0x0b as a request spacer

MUTATE OPTIONS	
1	Test all files with all root directories
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

TUNING OPTIONS	
1	Interesting File / Seen in logs
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)

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*.

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

XSS

Seguiremos un procedimiento muy similar al anterior para ver si el WAF se activa cuando se realiza un intento de **ataque XSS.** De esta forma, en lugar de pasar el ataque usando `GET` pondremos el *payload* del ataque como parámetro `POST` gracias a la opción `-d` del comando `curl`. Para activar el WAF pasaremos una cadena de prueba XSS típica: `<script>alert('test')</alert>` con `curl <proxy IP>/-d "<script>alert('test')</script>"`

Inyección SQL

Al igual que las pruebas XSS, realizamos una **prueba de detección de ataques de inyección SQL** pasando como parámetro `POST` a los usuarios de la cadena `DROP DATABASE;`. Comprueba también que solo identifica la sintaxis SQL correcta introduciendo errores tipográficos en la cadena proporcionada para ver si la captura o no.

Bloque 3: Herramientas AST para el desarrollo de aplicaciones

Ejercicio L10B3_SASTSCA: *Herramienta de detección automatizada de vulnerabilidades en aplicaciones SonarQube* (SAST, con módulo SCA opcional)

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

Otra información necesaria para su realización

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.

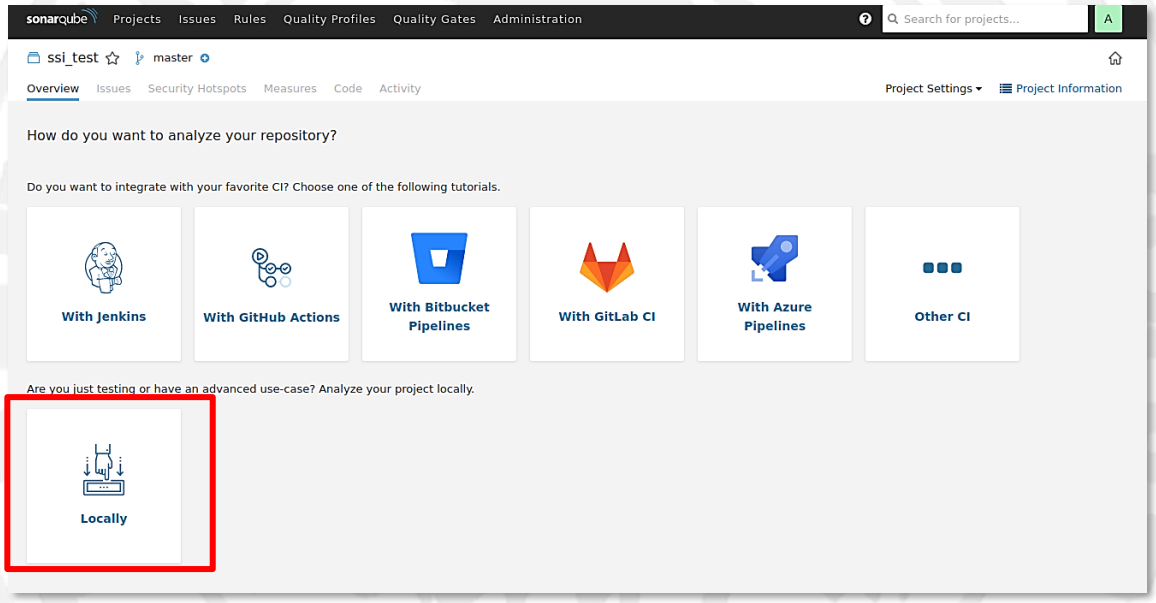
🔍 ***Admite alrededor de 20 lenguajes de programación diferentes (principalmente Java o C#) y busca bugs, vulnerabilidades, security hotspots, code smells, métricas de calidad de código, etc.***

Se integra con *Maven*, *Gradle*, y otras plataformas de construcción de aplicaciones también utilizadas en otros cursos como **ASW** 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.

SAST automatizado con *SonarQube*

Esta herramienta puede identificar errores que no se han detectado en nuestra revisión manual y pruebas. Este es el objetivo de la asignatura **IPS**, por lo que en este laboratorio nos centraremos en encontrar **vulnerabilidades** que puedan estar presentes en el código fuente. Para usar *SonarQube* necesitamos seguir estos pasos desde tu máquina virtual ***Ubuntu*:**

- Elegir un proyecto en un lenguaje soportado para analizar (preferiblemente en *Java* o *C#*, usando *Maven* o *Gradle*). Si no tienes uno propio, puedes descargar el archivo zip de este repositorio (https://github.com/appsecco/dvja) para probar cómo funciona esta herramienta. Esta zip también se proporciona como parte de los materiales del **laboratorio 10.** Descomprime el archivo en una carpeta a la que puedas acceder luego de la máquina virtual. A partir de ahora asumiremos que vamos a usar este proyecto de prueba, si usas uno propio debes adaptar las instrucciones a tu caso.
- Instalar *Maven* **en la máquina virtual de *Ubuntu*** (`sudo apt install maven`)
- Ejecuta un contenedor *SonarQube* ejecutando el *script* `run_sonarqube.sh` que incluimos en los **materiales del laboratorio 10.** Descarga 0.5Gb de información y tarda algún tiempo en completarse.
- Inicia sesión en http://localhost:9000 con las 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*.



  - Mira el token generado y haz clic en continuar. Luego, elije ***Maven***. Esto se debe a que nuestro proyecto de prueba se construye con *Maven* pero, como ves, hay muchas otras opciones que puedes usar en el futuro (como por ejemplo, tu TFG 😉)
  - **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 que descomprimiste 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 esto.

```
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 ***IPS*** o ***CPM***. Las categorías más vinculadas con nuestra asignatura son ***Vulnerabilities*** y la pestaña ***Security Hotspots***.

- Abra la pestaña ***Security Hotspots*** y mira los 7 posibles problemas de los que informa. Responde a estas tres preguntas con la información que proporciona la herramienta:
  1. *¿Crees que el problema de seguridad de Authentication es real o un falso positivo?*
  2. De acuerdo con la descripción de las vulnerabilidades de inyección SQL dada en el **Tema 6** de teoría**,** *¿crees que los problemas de seguridad relacionados identificados por la herramienta pueden ser potencialmente peligrosos?*
  3. De acuerdo con lo que vimos en el **Tema 2** de teoría *(criptografía), ¿crees que la aplicación está utilizando un método de hashing fuerte?*

SCA automatizado con *SonarQube*

🔍 ***NOTA:*** ***Si te es posible, intenta aumentar la RAM de la máquina virtual a al menos 4 Gb para este ejercicio. Se puede hacer con la máquina virtual por defecto (2 Gb), pero agotará la RAM de la máquina virtual durante el proceso y se bloqueará después de generar el informe, por lo que tendrás que reiniciarla. En la siguiente descripción se supone que no puedes aumentar la RAM.***

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://owasp.org/www-project-dependency-check/. Para ello, sigue estos pasos:

- Apaga la máquina virtual y, si es posible, aumenta su RAM.
- Vuelve a arrancar la máquina virtual e inicia sesión como lo haces normalmente. Localiza el archivo `pom.xml` de nuestro proyecto de prueba (en la carpeta en la que lo descomprimiste antes), ábrelo y asegúrate de añadir las líneas que aparecen en rojo aquí:

```
                …
        <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>
                <version>6.5.2</version>
                <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`
- Analizar el informe: *¿hay dependencias vulnerables? ¿Se puede obtener información sobre cada vulnerabilidad?* (descripción, puntuación...)
- Apaga la máquina virtual de nuevo y restaura la RAM a su valor original si quieres (y si la cambiaste).

🛠 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 quieres 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?*

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.

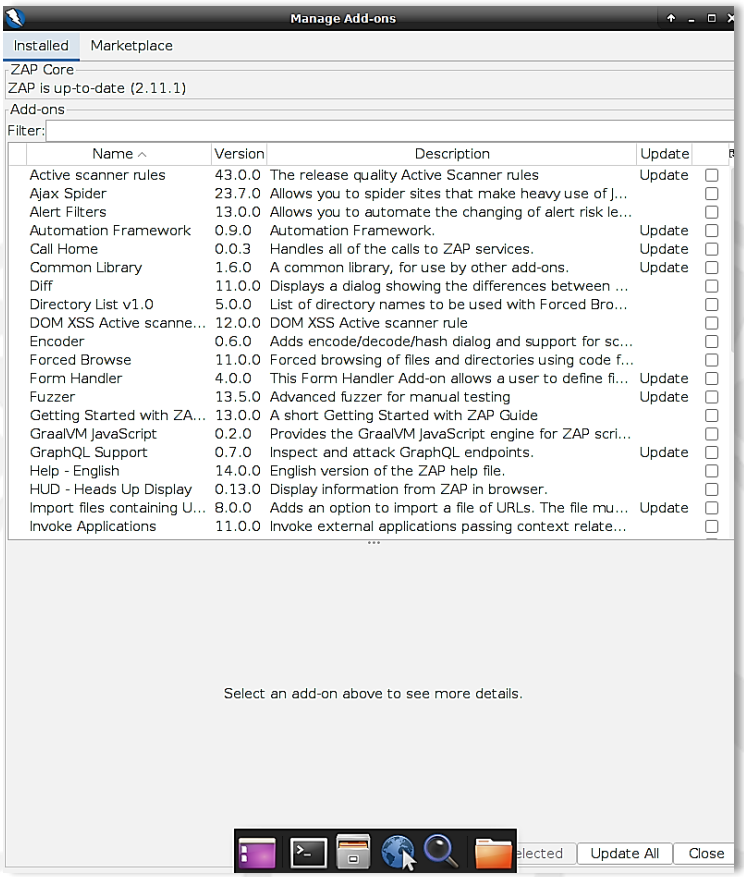
Introducción

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.***

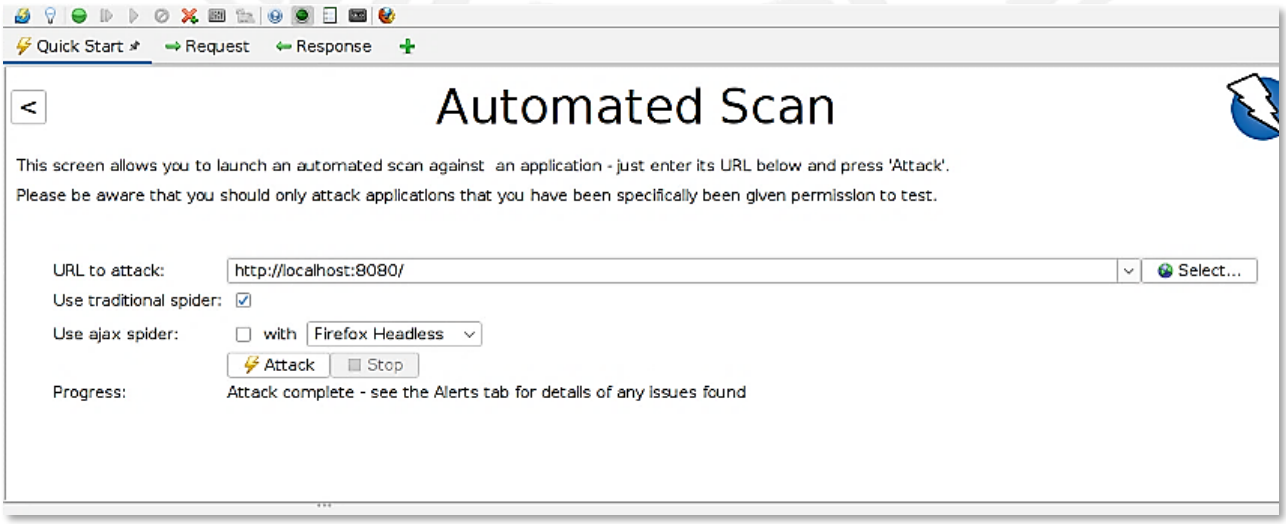
Lo cierto es que para poder probar este DAST **necesitaremos pues una aplicación "víctima"**, y podemos usar cualquiera de otra asignatura para ello que tengamos a mano y de la que **seamos propietarios o hayamos logrado una autorización por escrito**. Si no fuera el caso, una buena forma de probarla 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 8080:80 vulnerables/web-dvwa` (aunque **es mejor poner otro puerto**, porque el `8080` choca con ZAP).

Hecho esto, podemos instalar ya la herramienta según el SO usado para hacer las pruebas siguiendo estas indicaciones: https://software.opensuse.org/download.html?project=home%3Acabelo&package=owasp-zap. **Se recomienda hacerlo desde los repositorios del sistema operativo**. Hecho esto, ejecutamos la herramienta con `owasp-zap` y dejamos que se actualicen todos los *add-ons* que vienen de serie para lograr el mejor funcionamiento posible (*Update All*).



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.

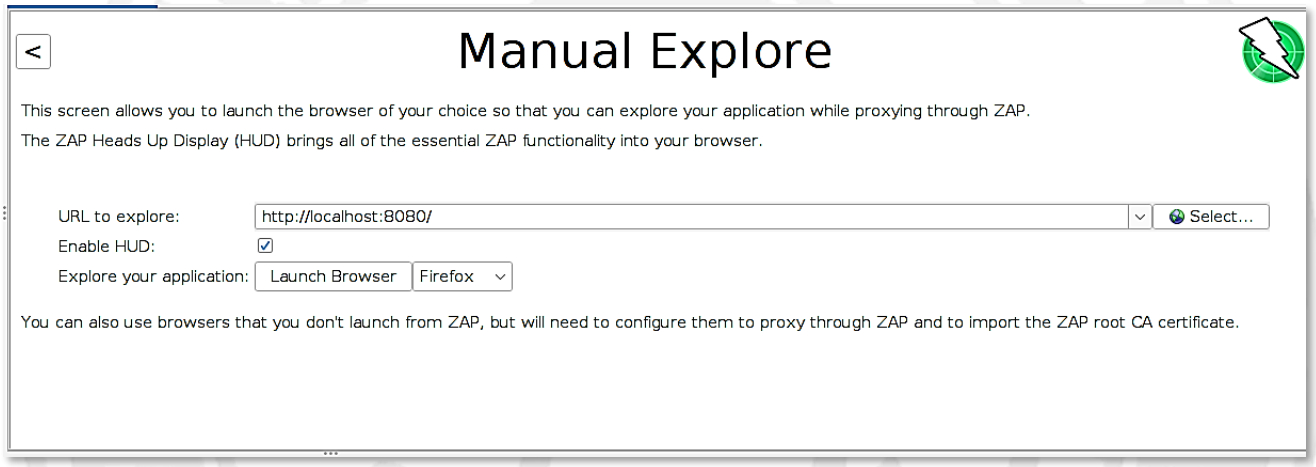


Si todo ha ido correctamente, reconoceremos algunos de los problemas que hemos descrito en la teoría, a los que tendríamos que buscar solución antes de poner la aplicación en producción.

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.***

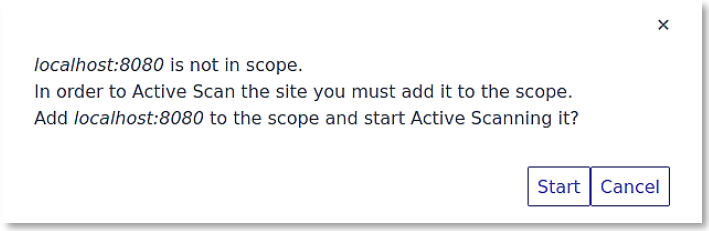


Hay una cosa que debemos tener en cuenta sobre el navegador que nos muestra la herramienta: es el navegador predeterminado del sistema con un plugin llamado **HUD**, que nos saca **en el lado izquierdo las vulnerabilidades de la página web** que estamos visitando en esos momentos (con su gravedad con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

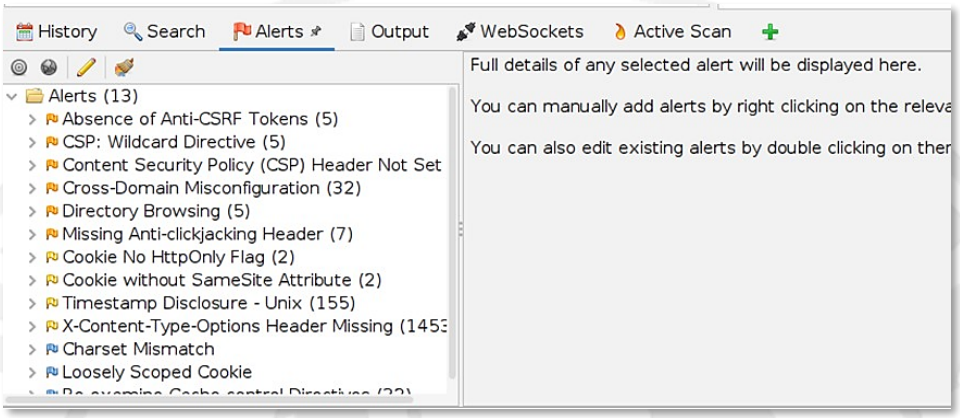
🔍 ***Como veremos, además ZAP sigue haciendo su trabajo "por debajo", tal y como lo hacía con el escaneo activo.***

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 podremos ver sobre los propios iconos del HUD.



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



- **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 olvidar 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 desees y tomar notas extendidas de tus actividades para crear un log de lo que has hecho. Recuerda que el material que elabores 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: *¿Un proxy inverso realmente sirve sitios web por sí mismo?*	
	Entiendes cuál es la funcionalidad principal de un WAF	
	Sabes cómo iniciar escaneos web usando `nikto`	
	Puedes verificar si *ModSecurity* 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: *¿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?*	
	Puede responder a estas preguntas: *¿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?*	
	Puede responder a estas preguntas: *¿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?*	
	Puedes responder a estas preguntas: *¿Se registran las solicitudes bloqueadas de ModSecurity? En ese caso, ¿en qué fichero?*	
	Puede responder a esta pregunta: *¿Protege un WAF contra los intentos de ataque automatizados?*	
	Puede responder a esta pregunta: *¿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?*	
	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: *¿Crees que este tipo de herramientas localizan todos los errores posibles? Si no es así, ¿crees que conviene usarlas?*	
	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: *¿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?*	
	Puedes responder a esta pregunta: *¿Crees que los WAF proporcionan la máxima protección contra ataques, o deberían tratarse como una capa adicional de protección?*	
	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 prevenidas por una herramienta de análisis estático como *SonarQube*	
	**¡Hold the door!** Puedes proteger un servidor web mediante un proxy inverso securizado con un WAF y con capacidades de detección de intrusiones, y localizar vulnerabilidades en el código o sus dependencias mediante la implementación de herramientas SCA y SAST automatizadas	