Implatación de técnicas y herramientas de pentesting en el proceso de

desarrollo de software

**Borrador**

Índice

[1. Introducción 4](#_Toc77068523)

[1.1. Motivación 4](#_Toc77068524)

[1.2. Objetivos 4](#_Toc77068525)

[1.3. Metodología y Competencias 5](#_Toc77068526)

[1.4. Estructura del proyecto 5](#_Toc77068527)

[2. Análisis del estado del arte 6](#_Toc77068528)

[2.1. Proceso de pentesting 6](#_Toc77068529)

[2.1.1. ¿Qué es una prueba de penetración o pentest? 6](#_Toc77068530)

[2.1.2. Fases de la prueba de intrusión 7](#_Toc77068531)

[2.2. Detalle y Clasificación de vulnerabilidades OWASP Top 10 9](#_Toc77068532)

[2.2.1. A1:2017 - Inyecciones 11](#_Toc77068533)

[2.2.2. A2:2017 - Pérdida de autenticación y gestión de sesiones (Broken Authentication) 13](#_Toc77068534)

[2.2.3. A3:2017 - Exposición de datos sensibles (Sensitive Data Exposure) 15](#_Toc77068535)

[2.2.4. A4:2017 - XML External Entities (XXE) 15](#_Toc77068536)

[2.2.5. A5:2017 - Pérdida de control de acceso (Broken Access Control) 17](#_Toc77068537)

[2.2.6. A6:2017 - Configuración de seguridad incorrecta (Security Misconfiguration) 17](#_Toc77068538)

[2.2.7. A7:2017 - Secuencia de comando de sitios cruzados (XSS) 18](#_Toc77068539)

[2.2.8. A8:2017 - Deserialización insegura (Insecure Deserialization) 19](#_Toc77068540)

[2.2.9. A9:2017 - Uso de componentes con vulnerabilidades conocidas 20](#_Toc77068541)

[2.2.10. A10:2017 - Registro y monitoreo insuficientes 20](#_Toc77068542)

[2.3. Herramientas Análisis de código. 21](#_Toc77068543)

[2.3.1. Herramientas Análisis estático de código. 22](#_Toc77068544)

[2.3.2. Herramientas Análisis dinámico de código. 25](#_Toc77068545)

[3. Diseño solución técnica 26](#_Toc77068546)

[3.1. Metodología de pruebas 26](#_Toc77068547)

[3.1.1. Alcance y términos de la prueba de intrusión. 27](#_Toc77068548)

[3.1.2. Recolección de información. 28](#_Toc77068549)

[3.1.3. Generación reporte análisis estático de código 29](#_Toc77068550)

[3.1.4. Análisis de vulnerabilidades. 31](#_Toc77068551)

[3.1.5. Generación de Informes 33](#_Toc77068552)

[3.2. Infraestructura de pruebas 34](#_Toc77068553)

[4. Ejecución casos de prueba 36](#_Toc77068554)

[4.1. Aplicación en desarrollo de aplicaciones Web 36](#_Toc77068555)

[4.1.1. WebGoat 37](#_Toc77068556)

[4.1.2. Damn Vulnerable Web application 41](#_Toc77068557)

[4.1.3. Juice Shop 45](#_Toc77068558)

[4.1.4. WebGoat.Net 48](#_Toc77068559)

[5. Conclusiones y trabajo futuro 50](#_Toc77068560)

[6. Anexos 51](#_Toc77068561)

[6.1. Glosario de términos 51](#_Toc77068562)

[6.2. Documentos relacionados 53](#_Toc77068563)

[6.3. Instalación Software Necesario 53](#_Toc77068564)

[6.4. Instalación SonarQube 56](#_Toc77068565)

[6.4.1. Actualizar paquetes distro 56](#_Toc77068566)

[6.4.2. Instalar y configurar PostgreSQL 56](#_Toc77068567)

[6.4.3. Instalación Configuración de SonarQube 57](#_Toc77068568)

1. Introducción

La ciberseguridad es un derivado de la seguridad TI. Involucra todos los procesos y técnicas utilizadas para proteger la información digital alojada en computadoras, servidores y redes del acceso no autorizado, ataques o su destrucción.

A diferencia de la seguridad TI, que protege tanto la información física como digital, la ciberseguridad solo se enfoca en proteger la información digital.

En los últimos tiempos el desarrollo de software ha acortado los tiempos de desarrollo, pasando del ciclo clásico de desarrollo en cascada con un tiempo de desarrollo de meses a un ciclo de desarrollo ágil con un tiempo de desarrollo de semanas. Este acortamiento de los tiempos de desarrollo ha llevado apareados cambios en todo los procesos involucrados en el proceso de desarrollo del software incluidos los procesos de pruebas de seguridad.

Además, el acortamiento de los tiempos de desarrollo también ha provocado la necesidad de una mayor conexión ente los distintos miembros involucrados en los distintos procesos de desarrollo del software.

Es por ello por lo que en el presente PFG trato de describir un proceso de pentesting que ayude a pentester a reducir el tiempo necesario en el desarrollo de la pruebas de seguridad necesarias para poder desplegar un proyecto en producción en el menor tiempo posible.

Además de generar mecanismos de comunicación con el equipo de desarrollo enfocados en resolver los problemas que detecten dichas pruebas de seguridad en el menor tiempo posible.

* 1. Motivación

En el tiempo que llevo involucrado en proyectos de la ciberseguridad he visto muchas herramientas y metodologías que abordan el proceso de pruebas de seguridad o pruebas de pentesting.

Normalmente, el proceso de pentesting es un proceso manual y muy dependiente de la persona que realice las mismas y muchas veces los reportes presentado solo se dedican a señalar el problema y no suelen enfocarse a la solución de los problemas.

Es por ello por lo que el presente proyecto fin de grado quería plasmar un proceso de pentesting que sirva de punto de partida para la realización de pruebas de seguridad al pentester y que se encuentre alineado con el ciclo de desarrollo software y enfocado a la solución de los problemas que se detecten en la evaluación de un software.

* 1. Objetivos

Durante la realización de este TFG se pretende realizar una breve introducción al proceso de pentesting, la fase que involucran dicho proceso, las herramientas que se pueden utilizar para llevar a cabo dicho proceso, así como la descripción de los fallos mas comunes que se suelen encontrar en las evaluaciones y detallar los documentos que se deben generar enfocados a tener una comunicación con el equipo de desarrollo encaminados a la solución de los defectos y a la mejora de la calidad del software evaluado.

Para poder lograr lo que se ha indicado en el párrafo anterior, se plantean los siguientes objetivos:

* Detalle del proceso de pentesting y sus fases.
* Detallar los defectos de seguridad más comunes que se suelen encontrar en la ejecución de pruebas de seguridad o pentesting.
* Detallar el funcionamiento de las herramientas que se suelen utilizar en la ejecución del proceso de pentesting
* Detallar la realización de los análisis de código estático y dinámicos de aplicaciones que hagan uso de las tecnologías más comunes en la web.
* Desarrollo de una pequeña herramienta que facilite la realización de los reportes de análisis estáticos de código enfocados a la comunicación con el equipo de desarrollo de cara a la solución de los defectos encontrados y a la mejora de a calidad del software analizado.
* Ejecución del proceso de pentesting sobre aplicaciones que usen las tecnologías de desarrollo más comunes en el desarrollo de aplicaciones web.

* 1. Metodología y Competencias

Para la realización del PFG haremos uso de la metodología [OWASP Application Security Verification Standard (ASVS) 4.0](https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf) que proporciona una base para probar los controles técnicos de seguridad de las aplicaciones web.

Las competencias que se tienen que seguir para la correcta realización del proyecto son:

**[CO1]** Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

**[IC6]** Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.

**[TI7]** Capacidad para comprender, aplicar y gestionar la garantía y seguridad de los sistemas informáticos.

* 1. Estructura del proyecto

El presente PFG está estructurado en cinco capítulos.

En el capítulo primero, se expone la introducción, los objetivos, la metodología, la estructura, etc.

En el capítulo dos estará enfocado al análisis del estado del arte, donde expondremos en que consiste el proceso de pentesting y sus fases. También se detallarán los defectos de seguridad más comunes encontrados en los procesos de pentesting además de describir a nivel teórico los procesos de análisis estático y dinámico de código.

El capítulo tres estará enfocado al diseño de la solución técnica para llevar a cabo el proceso de pentesting.

En capítulo cuatro se hará uso del proceso definido en el capitulo tres sobre aplicaciones desarrolladas con las tecnologías más comunes en el desarrollo de aplicaciones web.

En el capítulo seis se expondrán las conclusiones del proyecto, así como posibles trabajos futuros sobre este tema.

También se incluye un anexo donde se detallarán los informes generados en la realización del proceso de pentesting

1. Análisis del estado del arte
   1. Proceso de pentesting
      1. ¿Qué es una prueba de penetración o pentest?

Según la definición de OWASP una prueba de penetración o pentesting, a veces denominado prueba de caja negra, es esencialmente el arte de probar un sistema o aplicación para descubrir vulnerabilidades de seguridad, sin conocer el funcionamiento interno de la mismas. Normalmente el equipo encargado de las pruebas de penetración accede a las aplicaciones como si fuesen usuarios. El pentester tratará con que ese nivel de acceso encontrar vulnerabilidades que se puedan explotar en la aplicación.

El propósito de la prueba de penetración es determinar la presencia de vulnerabilidades potencialmente explotables y analizar el impacto de estas, sí se detecta alguna. La mejor forma de probar una defensa es tratando de penetrar en ella.

* + 1. Fases de la prueba de intrusión

A la hora de realizar una prueba de intrusión o pentest distinguimos las siguientes fases, basándonos en la distinción realizada en pentesting con Kali {ref1}, dichas fases son las siguientes:

* Alcance y términos de la prueba de intrusión.
* Recolección de información.
* Análisis de vulnerabilidades.
* Explotación de vulnerabilidades.
* Postexplotación del sistema.
* Generación de informes.

***Alcance y términos de la prueba de intrusión.***

Para esta fase normalmente se genera un documento de plan de pruebas. En muchos casos es necesaria la revisión y aprobación de dicho documento por parte del dueño del sistema a probar (SUT) antes de poder comenzar con el proceso de pentesting.

En dicho documento de pruebas se suele detallar la siguiente información:

* Sistema sobre el que se realizan las pruebas
* Los tipos de prueba a realizar.
* Herramientas que se van a utilizar.
* Proceso de seguimiento de los defectos encontrados.
* Documentos que se entregarán durante el proceso de pentesting.
* Restricciones en la ejecución de la prueba de intrusión

***Recolección de información.***

Una vez definido el plan de pruebas procederemos a recolectar información del sistema o aplicación indicado en dicho plan. Principalmente obtendremos información mediante los procesos de enumeración y análisis de código que detallaremos en el siguiente capítulo.

***Análisis de vulnerabilidades.***

Al finalizar los procesos anteriores se analizarán los defectos encontrados para descartar falsos positivos y después se hará entrega un reporte de análisis dinámico con los defectos no descartados. Para cada uno de los defectos detectados que se incluyan en el reporte abriremos defecto en el sistema de gestión de defectos.

***Explotación de vulnerabilidades.***

En el caso de que uno o varios defectos necesiten ser explotados, y siempre solicitando permiso se detallará el proceso de explotación indicando las herramientas y exploits necesarios para realizar este proceso. En este proceso también se deben detallar las consecuencias, si las hubiese de la ejecución de las herramientas y exploits a utilizar sobre la aplicación o sistema objetivo.

***Postexplotación del sistema.***

En este caso también es necesario solicitar permiso al dueño del sistema, detallando la forma en que persistirá el ataque en la aplicación o sistema objetivo.

***Generación de informes.***

Llegados a este punto ya se deben haber hecho entrega de los reportes del análisis estático, si se dispone de acceso al código fuente, y del reporte de análisis dinámico. S ́ı se ejecutasen los procesos de explotación o Postexplotación se ampliaría el reporte de análisis dinámico con la información recabada en dichos procesos.

A parte de los reportes anteriores, se debe entregar un informe de resultado de pruebas con el resultado de ejecución del proceso de pentest incluyendo en el mismo el detalle de los defectos reportados en el sistema de gestión de defectos, si es posible, así como el estado en que se encuentran en el momento de entrega de dicho reporte.

* 1. Detalle y Clasificación de vulnerabilidades OWASP Top 10

En índice de vulnerabilidades web OWASP Top 10, [versión 2017](https://wiki.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf),clasifica los vulnerabilidades más comunes encontradasen los datos de aportados por cientos de organizaciones y más de 100.000 aplicaciones y servicios web del mundo real.

En su última versión las vulnerabilidades más comunes encontradas fueron las siguientes:

|  |
| --- |
| **OWASP Top 10 - 2017** |
| [A1:2017- Inyecciones](https://owasp.org/www-project-top-ten/2017/A1_2017-Injection) |
| [A2:2017- Pérdida de autenticación y gestión de sesiones (Broken Authentication)](https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication) |
| [A3:2017- Exposición de datos sensibles (Sensitive Data Exposure)](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure) |
| [A4:2017- Entidades Externas XML (XXE)](https://owasp.org/www-project-top-ten/2017/A4_2017-%20Entidades%20Externas%20XML%20(XXE)) |
| [A5:2017- Pérdida de control de acceso (Broken Access Control)](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control) |
| [A6:2017- Configuración de seguridad incorrecta (Security Misconfiguration)](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration) |
| [A7:2017- Secuencia de comando de sitios cruzados (XSS)](https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)) |
| [A8:2017- Deserialización insegura (Insecure Deserialization)](https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization) |
| [A9:2017- Uso de componentes con vulnerabilidades conocidas](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities) |
| [A10:2017- Registro y monitoreo insuficientes](https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring) |

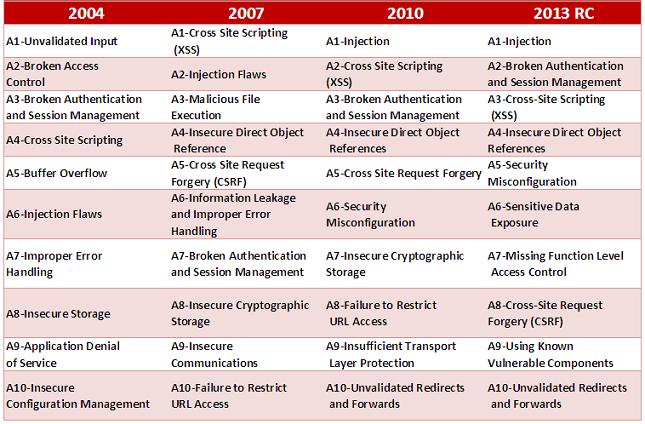


Ilustración 1 – Fuente [OWASP Top Ten 2013 RC: Evolución de las vulnerabilidades web | INCIBE-CERT](https://www.incibe-cert.es/blog/owasp-top-ten-2013rc-evolucion-vulnerabilidades)

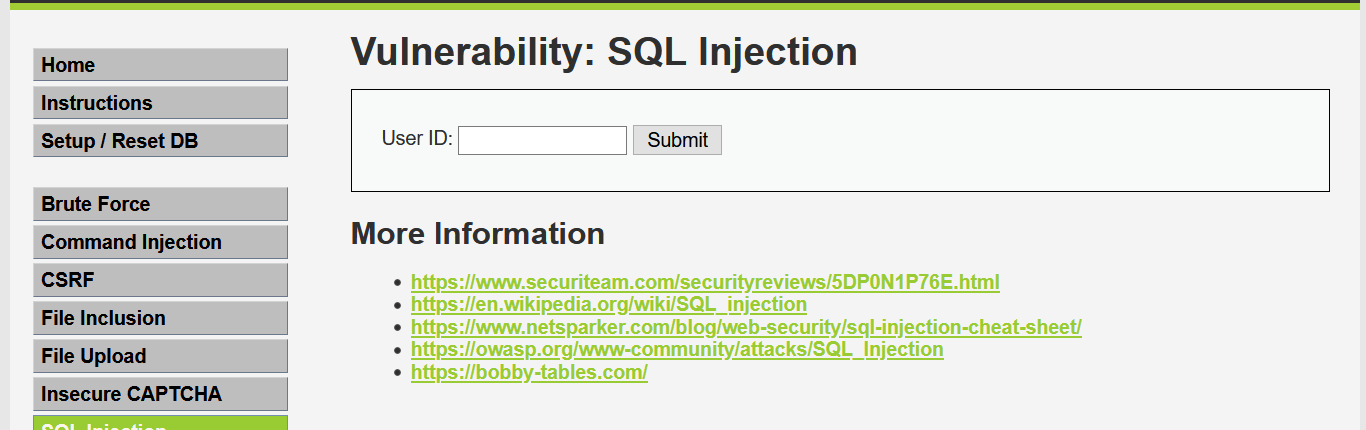
A continuación, detallamos en qué consisten cada una de las vulnerabilidades listadas en el OWASP top 10:

* + 1. A1:2017 - Inyecciones

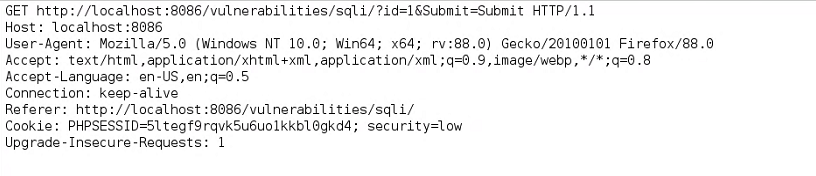
Las fallas de inyección, como SQL, NoSQL, comandos o LDAP ocurren cuando se envían datos no  
confiables a un intérprete, como parte de un comando o consulta. Los datos dañinos del atacante  
pueden engañar al intérprete para que ejecute comandos involuntarios o acceda a los datos sin  
la debida autorización.

La inyección de SQL (SQLi) es uno de los tipos de ataques de inyección de código más  
comunes y peligrosos, aprovechados por los atacantes con la intención de obtener  
información no autorizada o en sí generar problemas en los servidores de base de datos y  
comportamiento de aplicaciones.

Por Ejemplo, en la siguiente aplicación tenemos un formulario para mostrar la información de un usuario a partir de su identificador (ID):



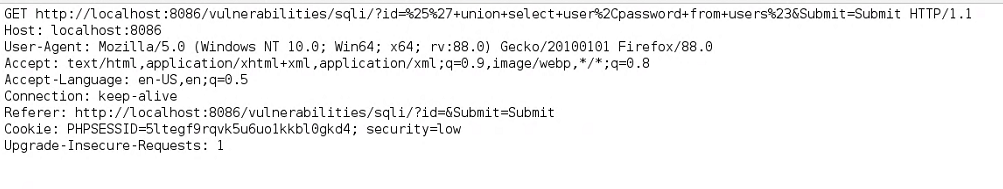
Un uso normal generaría este tipo de peticiones:



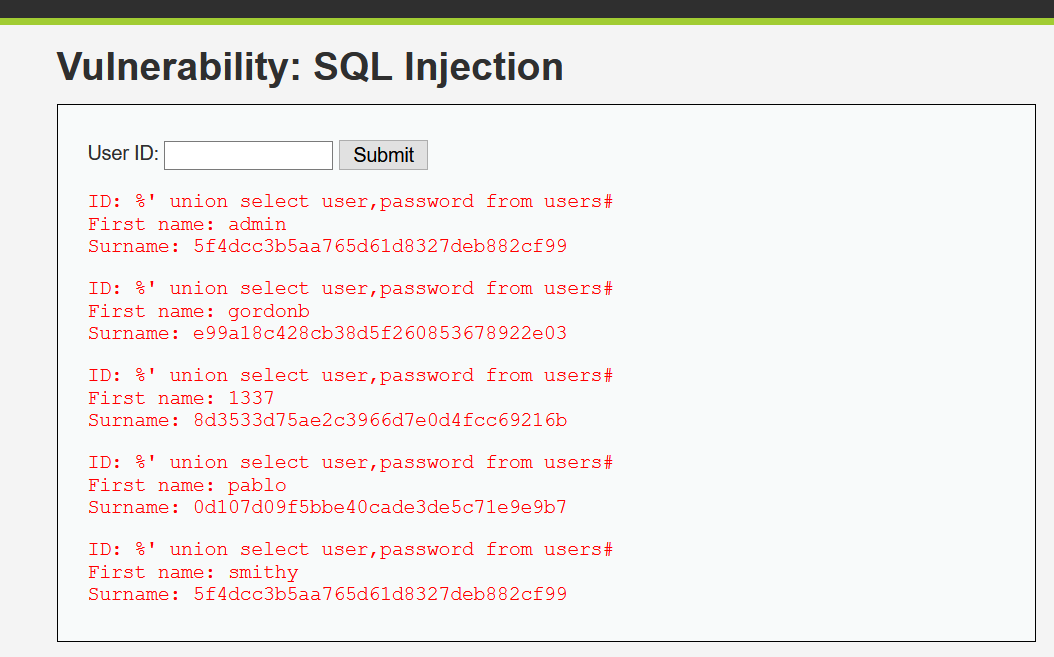
Pero si abusamos de la aplicación modificando el ID por la consulta:

%' union select user, password from users#

Se genera la siguiente petición:



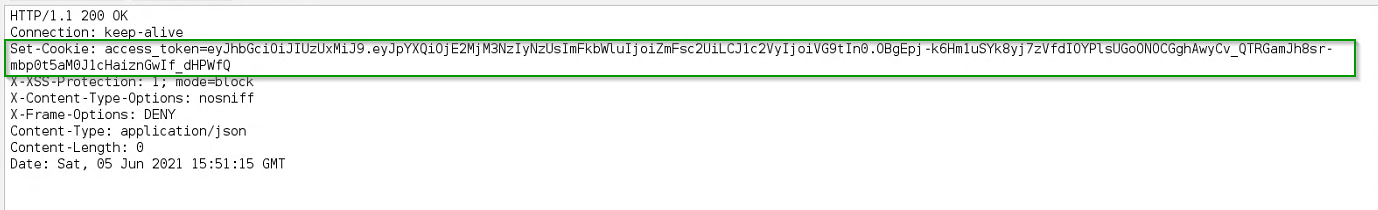
El resultado será que la aplicación nos devuelve todos los usuarios y password almacenados en la Base de datos:



* + 1. A2:2017 - Pérdida de autenticación y gestión de sesiones (Broken Authentication)

Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son  
implementadas incorrectamente, permitiendo a los atacantes comprometer usuarios y  
contraseñas, token de sesiones, o explotar otras fallas de implementación para asumir la  
identidad de otros usuarios (temporal o permanentemente).

En los últimos años se han detectado numerosas aplicaciones, sobre todo la que hacen uso de api para la gestión de los datos, que hacen uso de JSON Web Tokens (JWT) para la autenticación y autorización.



La captura de este token permite a los atacantes a realizar peticiones en nombre del usuario, puesto que, si decodificamos el token, podemos ver que identifica a un usuario concreto



Lo cual nos permite realizar cualquier petición en nombre del usuario haciendo uso de su token JWT



* + 1. [A3:2017 - Exposición de datos sensibles (Sensitive Data Exposure)](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure)

Muchas aplicaciones y servicios web no protegen adecuadamente datos sensibles, tales como  
información financiera, de salud o Información Personalmente Identificable (PII). Los atacantes  
pueden robar o modificar estos datos protegidos inadecuadamente para llevar a cabo fraudes  
con tarjetas de crédito, robos de identidad u otros delitos. Los datos sensibles requieren métodos  
de protección adicionales, como el cifrado en almacenamiento y tránsito.

* + 1. [A4:2017 - XML External Entities (XXE)](https://owasp.org/www-project-top-ten/2017/A4_2017-%20Entidades%20Externas%20XML%20(XXE))

Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos internos mediante la URI o archivos internos en servidores no actualizados, escanear puertos de la LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS).

Una entidad XML permite definir etiquetas que serán reemplazadas por contenido cuando se analice el documento XML. En general, existen tres tipos de entidades:

* Entidades internas
* Entidades externas
* Entidades parametrizadas.

Una entidad debe ser definida en el “Document Type Definition” (DTD), vemos un ejemplo:

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE data [<!ENTITY xxe SYSTEM "/etc/passwd">]>

<feed>

    <title>test</title>

    <description>test</description>

    <entry>

        <title>Hello</title>

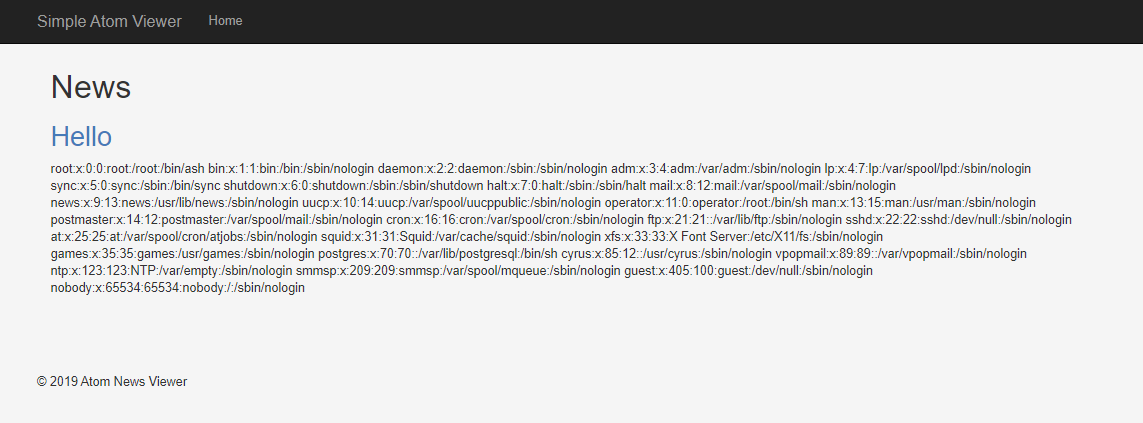
        <link href="http://example.com"></link>

        <content>&xxe;</content>

    </entry>

</feed>

En este caso el parser de XML carga la entidad externa **“SYSTEM”** que obtendrá el contenido del fichero **“/etc/passwd”** y devolverá el contenido de este fichero en la respuesta:



Por lo tanto, un ataque de entidad externa XML es un tipo de ataque contra una aplicación que analiza la entrada XML. Este ataque ocurre cuando la entrada XML que contiene una referencia a una entidad externa es procesada por un analizador XML configurado débilmente.

Este ataque puede conducir a la divulgación de datos confidenciales, denegación de servicio, falsificación de solicitudes del lado del servidor, escaneo de puertos desde la perspectiva de la máquina donde se encuentra el analizador y otros impactos del sistema.

* + 1. [A5:2017 - Pérdida de control de acceso (Broken Access Control)](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control)

Las restricciones sobre lo que los usuarios autenticados pueden hacer no se aplican  
correctamente. Los atacantes pueden explotar estos defectos para acceder, de forma no  
autorizada, a funcionalidades y/o datos, cuentas de otros usuarios, ver archivos sensibles,  
modificar datos, cambiar derechos de acceso y permisos, etc.

* + 1. [A6:2017 - Configuración de seguridad incorrecta (Security Misconfiguration)](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration)

La configuración de seguridad incorrecta es un problema muy común y se debe en parte a  
establecer la configuración de forma manual, ad hoc o por omisión (o directamente por la falta de  
configuración).

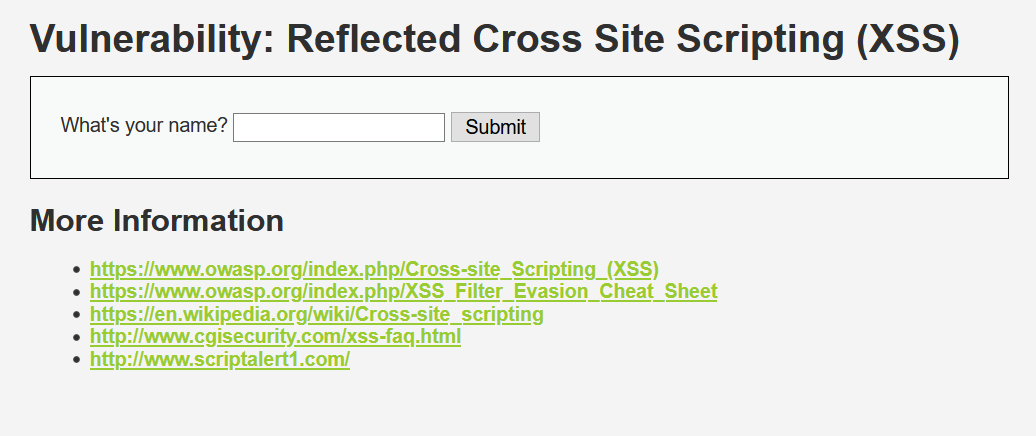
Son ejemplos: S3 buckets abiertos, cabeceras HTTP mal configuradas, mensajes  
de error con contenido sensible, falta de parches y actualizaciones, frameworks, dependencias y  
componentes desactualizados, etc.

* + 1. [A7:2017 - Secuencia de comando de sitios cruzados (XSS)](https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS))

Los XSS ocurren cuando una aplicación toma datos no confiables y los envía al navegador web  
sin una validación y codificación apropiada; o actualiza una página web existente con datos  
suministrados por el usuario utilizando una API que ejecuta JavaScript en el navegador.

Permiten ejecutar comandos en el navegador de la víctima y el atacante puede secuestrar una sesión, modificar (defacement) los sitios web, o redireccionar al usuario hacia un sitio malicioso.

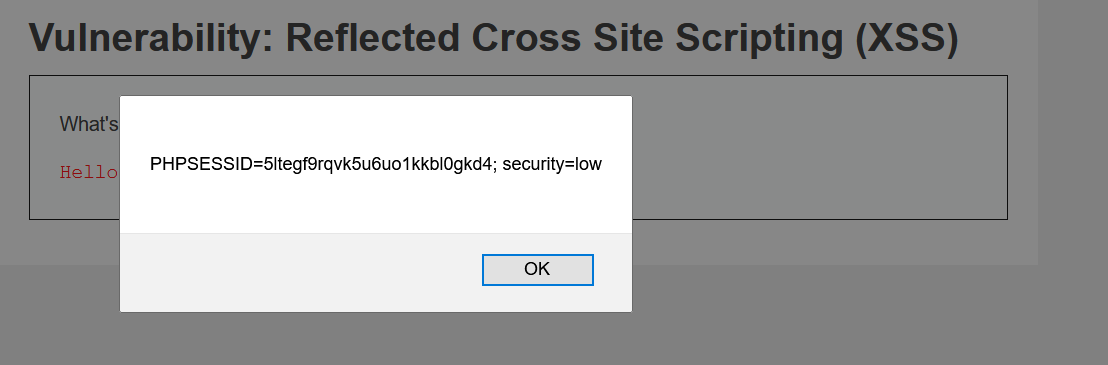
Por ejemplo, si tenemos una aplicación como la siguiente con un formulario de entrada de datos como el siguiente:



Si introducimos el siguiente script:

<script>alert(document.cookie)</script>

Vemos que al enviar el formulario se ejecuta el script en el navegador.



Podemos distinguir tres tipos de ataques XSS:

* **Reflejados:** Cuando el script malicioso está presente en la petición HTTP.
* **Almacenados:** El script malicioso es almacenado en el servidor, en la base de datos, en un fichero del sistema o cualquier otor objeto, y es visible cuando se muestra la página en el navegador.
* **Basados en el DOM:** Técnicamente se consideraría reflejado. Ocurre cuando el script malicioso incluye código html en la petición HTTP.
  + 1. [A8:2017 - Deserialización insegura (Insecure Deserialization)](https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization)

Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos  
pueden ser manipulados o borrados por el atacante para realizar ataques de repetición,  
inyecciones o elevar sus privilegios de ejecución. En el peor de los casos, la deserialización  
insegura puede conducir a la ejecución remota de código en el servidor.

La serialización es el proceso de convertir un objeto en un formato de datos que se puede restaurar más tarde. Las personas a menudo serializan objetos para guardarlos en el almacenamiento o para enviarlos como parte de las comunicaciones. La deserialización es lo contrario de ese proceso que toma datos estructurados de algún formato y los reconstruye en un objeto.

Hoy en día, el formato de datos más popular para serializar datos es JSON, no hace mucho el formato más común era XML.

Muchos lenguajes de programación ofrecen una capacidad nativa para serializar objetos. Estos formatos nativos suelen ofrecer más funciones que JSON o XML, incluida la personalización del proceso de serialización. Desafortunadamente, las características de estos mecanismos de deserialización nativos pueden reutilizarse para generar efectos maliciosos cuando se opera con datos que no son de confianza.

Se ha descubierto que los ataques de deserialización permiten ataques de denegación de servicio, control de acceso y ejecución remota de código. Los lenguajes de programación que se han conocido ataques de este tipo serian:

* PHP
* Python
* Ruby
* Java
* C/C++

Por ejemplo, este código Java aprovecha la serialización para codificar una tarea que detenga la aplicación durante 5 segundos:

import org.dummy.insecure.framework.VulnerableTaskHolder;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.util.Base64;

public class Serialize {

    public static void main(String[] args) throws IOException{

        var byteStream = new ByteArrayOutputStream();

        var objectStream = new ObjectOutputStream(byteStream);

        objectStream.writeObject(new VulnerableTaskHolder("myTask", "sleep 5"));

        String payload = Base64.getEncoder().encodeToString(byteStream.toByteArray());

        System.out.println(payload);

    }

}

Este código crea la tarea y la serializa generando el siguiente token:

rO0ABXNyADFvcmcuZHVtbXkuaW5zZWN1cmUuZnJhbWV3b3JrLlZ1bG5lcmFibGVUYXNrSG9sZGVyAAAAAAAAAAICAANMABZyZXF1ZXN0ZWRFeGVjdXRpb25UaW1ldAAZTGphdmEvdGltZS9Mb2NhbERhdGVUaW1lO0wACnRhc2tBY3Rpb250ABJMamF2YS9sYW5nL1N0cmluZztMAAh0YXNrTmFtZXEAfgACeHBzcgANamF2YS50aW1lLlNlcpVdhLobIkiyDAAAeHB3DgUAAAflBgYSCgML+WfAeHQAB3NsZWVwIDV0AAZteVRhc2s=

Dicho token al ser enviado en una petición al servidor provoca que la aplicación se detenga durante 5 segundos.

* + 1. [A9:2017 - Uso de componentes con vulnerabilidades conocidas](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities)

Los componentes como bibliotecas, frameworks y otros módulos se ejecutan con los mismos  
privilegios que la aplicación. Si se explota un componente vulnerable, el ataque puede provocar  
una pérdida de datos o tomar el control del servidor. Las aplicaciones y API que utilizan  
componentes con vulnerabilidades conocidas pueden debilitar las defensas de las aplicaciones y  
permitir diversos ataques e impactos.

* + 1. [A10:2017 - Registro y monitoreo insuficientes](https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring)

El registro y monitoreo insuficiente, junto a la falta de respuesta ante incidentes permiten a los  
atacantes mantener el ataque en el tiempo, pivotear a otros sistemas y manipular, extraer o  
destruir datos. Los estudios muestran que el tiempo de detección de una brecha de seguridad es  
mayor a 200 días, siendo típicamente detectado por terceros en lugar de por procesos internos.

* 1. Herramientas Análisis de código.

Dentro de los procesos actuales de [SSDLC](#Glosario) cada vez cobran más importancia la inclusión de herramientas de análisis de código durante el proceso de desarrollo del Software.

Dentro de los tipos de análisis de código, podemos hacer la siguiente distinción en base al tipo de análisis de código:

* ***Herramientas de análisis estático de código (***[***SAST***](#Glosario)***).***

El análisis estático es un proceso que se realiza sobre el código de una aplicación sin necesidad de ejecutarse.

El análisis de código estático, también conocido como Análisis de código fuente ([SCA](#Glosario)), realiza pruebas sobre el código fuente para la detección temprana de defectos en dicho código. El uso de este tipo de herramientas es recomendable realizarlos en la fase de implementación del ciclo de desarrollo seguro ([SSDLC](#Glosario))

* ***Herramientas de análisis dinámico de código (***[***DAST***](#Glosario)***).***

Este tipo de análisis se realiza sobre una aplicación o servicio desplegado y en ejecución, a diferencia del tipo anterior.

En análisis DAST enviará peticiones maliciosas al sistema objetivo para verificar la presencia de diversos tipos de ataques.

* ***Herramientas híbridas***

Las herramientas hibridas son aquellas que presentan proceso para definir los dos tipos de análisis anteriores.

* + 1. Herramientas Análisis estático de código.

La metodología OWASP ASVS 4.0 se introdujo una sección para añadir los controles de código fuente como un requisito más dentro de la lista de requerimientos para un desarrollo seguro:

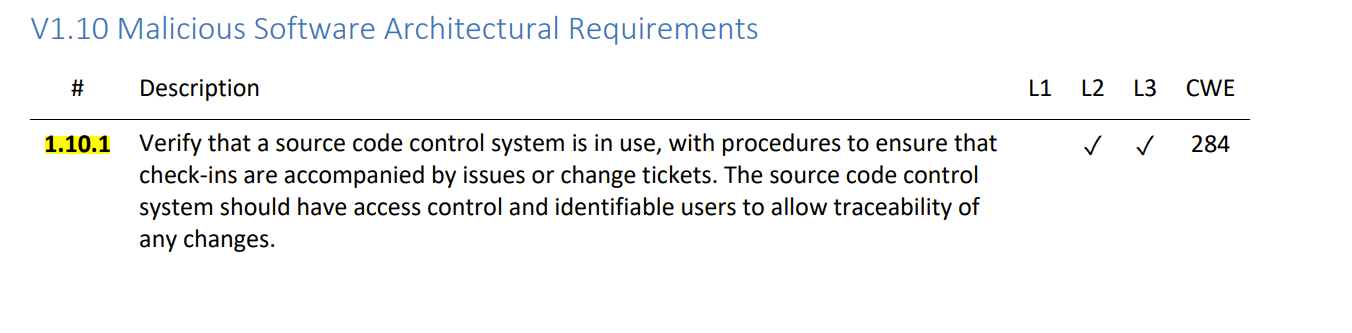


Ilustración 2 ASVS 4.0 10.0.1 Security Control

Actualmete en mucho de los ciclos de desarrollo estas herramientas se encuentran integradas dentro de los procesos de Integración continua (CI) y despliegue continuo (CD), esto permite que ante cualquier cambio en el código se ejecuten estas herramientas de forma automática permitiendo que ante cualquier cambio se ejecuten este tipo de herramientas de forma automática en los procesos de compilación y despliegue.

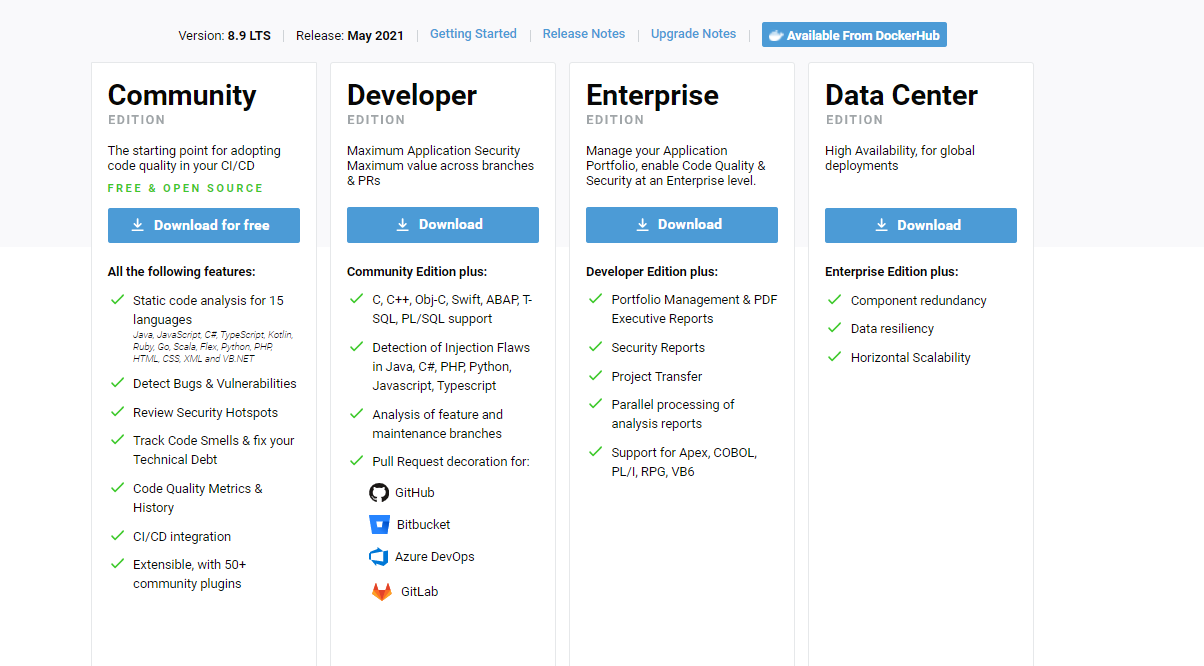
También es común que las herramientas de análisis de código estén integradas dentro de los IDEs de desarrollo; lo cual permite que los desarrolladores también puedan hacer uso de estas herramientas y mejorar la calidad del código antes de su entrega.

Entre las distintas herramientas de análisis, para la implementación de nuestra infraestructura de pruebas haremos uso de las siguientes herramientas:

* SonarQube
* Dependency-check

***SonarQube***

Es una plataforma de para el análisis estático de código, dispone de distintos escáneres para la mayor parte de lenguajes de programación. Entre las versiones disponibles de SonarQube, podemos hacer uso de la versión ***“Community”*** que es de uso libre.



La versión ***“Community”*** incluye escáneres para los siguientes lenguajes de programación:

* Java
* JavaScript
* C#
* TypeScript
* Kotlin
* Ryby
* Go
* Scala
* Flex
* Python
* PHP
* HTML
* CSS
* XML
* VB.Net

Además, mediante extensiones de la comunidad podemos añadir escáneres para los siguientes lenguajes:

* PL\SQL
* C\C++

La principal razón para hacer uso de SonarQube es que, con el paso del tiempo, se ha convertido en el estándar de facto de las herramientas de análisis estático de código debido, en gran medida, a las siguientes características:

* Pose escáneres de análisis estático de código para la mayoría de los leguajes de desarrollo utilizados.
* Es fácil de extender a nuevos lenguajes
* Es fácil de integrar en procesos CI/CD
* Dispone de integraciones con otras herramientas de análisis de código

[***Dependency-check***](https://owasp.org/www-project-dependency-check/)

Es una herramienta de análisis de dependencias que intenta detectar vulnerabilidades divulgadas públicamente contenidas en las dependencias de un proyecto. Para ello, determina si existe un identificador de enumeración de plataforma común (CPE) para una dependencia determinada. Si lo encuentra, generará un informe vinculado a las entradas [CVE](#Glosario) asociadas.

Actualmente OWASP Dependency-Check puede analizar dependencias de proyectos Java y .Net, que se encuentran totalmente soportados otros lenguajes como Ruby, Node.js, PHP (composer), Swift Package Manager y Python tienen un soporte más limitado.

El componente de análisis de dependencias de OWASP Dependency-Check puede ser ejecutado de las siguientes formas:

* Ant task
* [Command Line Tool](https://github.com/jeremylong/DependencyCheck/releases/download/v6.1.6/dependency-check-6.1.6-release.zip)
* Grandle plugin
* [Maven plugin](https://search.maven.org/#artifactdetails%7Corg.owasp%7Cdependency-check-maven%7C6.1.6%7Cmaven-plugin)
* SBT plugin

El uso de dependency-check desde la línea de comandos tiene los siguientes parámetros principales:

|  |  |
| --- | --- |
| **Parámetro** | **Descripción** |
| **--project** | Especifica el nombre del proyecto que aparecerá en el reporte. |
| **--scan** | Directorio donde se encuentran las librerías de terceros. |
| **--out** | Directorio de salida del reporte de análisis de dependencias. |
| **--suppresion** | Fichero .xml que contiene vulnerabilidades que deben de ser excluidas del reporte (falsos positivos) |

Ejemplo de uso:

  dependency-check.bat

      --project "juice-shop"

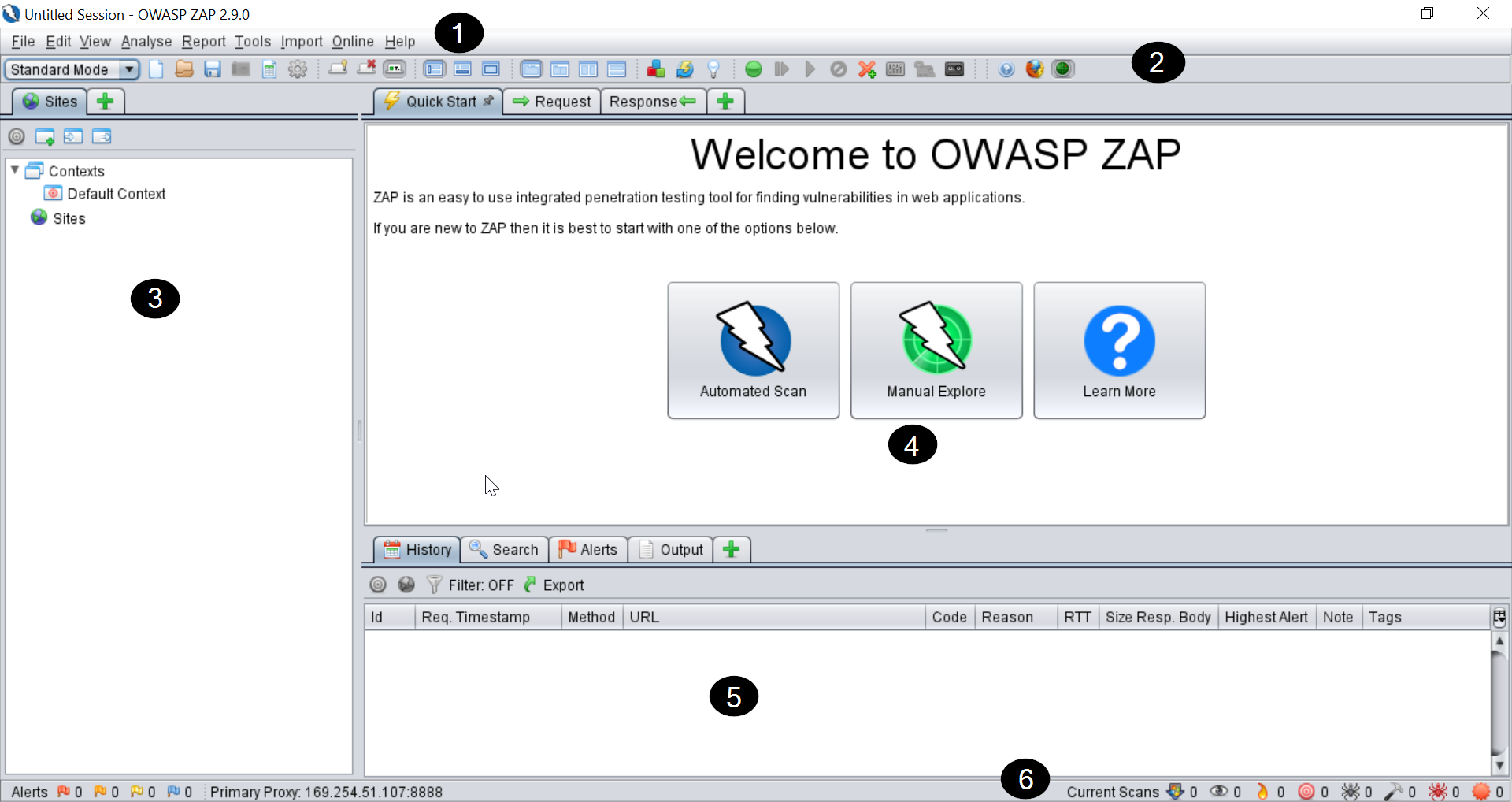
      --scan "D:\CodigoAnalisis\Seguridad\juice-shop\node\_modules"

      --out "D:\CodigoAnalisis\Seguridad\WebGoat.NET\reports"

* + 1. Herramientas Análisis dinámico de código.

Para las ejecuciones de análisis dinámico haremos uso de la herramienta Zed Attack Proxy (ZAP) de OWASP en su versión 2.10. OWASP Zap es una de las herramientas de software para análisis dinámico de aplicaciones que es mantenida y distribuida por la organización [OWASP](https://owasp.org/www-project-zap/). Su principal objetivo es el análisis de seguridades en aplicaciones web orientados a empresas, se caracteriza por ser de código abierto y totalmente gratuita.

La Interfaz de OWASP ZAP Desktop está compuesta de los siguientes elementos:



**1. Barra menú** – Proporciona acceso a las funcionalidades manuales y automáticas de la aplicación.  
**2. Barra herramientas** – Incluye botones de acceso rápido a las funciones más comunes.  
**3. Panel vista árbol –** Muestra los sitios visitados, así como los scripts utilizados.  
**4. Espacio de trabajo** – Muestra la peticiones y respuestas de las peticiones y permite editarlas.  
**5. Ventana de información** – Muestra los detalles de las herramientas automáticas y manuales utilizadas.   
**6. Pie** – Muestra el resumen de alertas encontradas por los distintos escáneres realizados.

Para más información consultar documentación, [ver documentación ZAP UI](https://www.zaproxy.org/docs/desktop/ui/)

A la hora de ejecutar el análisis dinámico haremos uso de las siguientes políticas de pruebas que serán ejecutados en cada una de las iteraciones para cada aplicación o sistema objetivo:

* [***Escáner regular:***](#DocumentosRelaconados) Para ampliar las rutas válidas dentro de los dominios a evaluar más allá de la utilizadas en sesión de pruebas utilizada.
* [***Escáner completo:***](#DocumentosRelaconados) A partir del resultado del escáner regular, donde se ampliará la batería de pruebas a realizar.

1. Diseño solución técnica
   1. Metodología de pruebas

Como metodología de pruebas para el proceso haremos uso de [OWASP Application Security Verification Standard (ASVS) 4.0](https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf) que proporciona una base para probar los controles técnicos de seguridad de las aplicaciones web. El proyecto clasifica los distintos controles en tres niveles. En este caso cubriremos todos los controles incluidos en el **nivel 2.**

Para abordar el proceso de pentesting los dividiremos en las fases definidas en el apartado 2.1.2, para la ejecución del proceso de pentesting ejecutaremos todas las fases menos la de explotación y postexplotación.

***Alcance y términos de la prueba de intrusión.***

Para cada una de las aplicaciones crearemos un documento definición del plan de pruebas de seguridad donde se detallará toda la información de las pruebas de seguridad a ejecutar.

***Recolección de información.***

En esta fase ejecutaremos el análisis estático de dependencias y generaremos un reporte del análisis estático de código. Los resultados del análisis estático servirán como base para crear un plan de pruebas para el análisis dinámico.

***Análisis de vulnerabilidades.***

En esta fase ejecutaremos el análisis dinámico de código a partir del plan de pruebas generado con la información obtenida en la fase anterior.

En esta fase ejecutarán dos veces el escáner de análisis dinámico con distinto número de reglas:

* **Escáner regular:** Para descubrir todas las posibles rutas validados debajo de los dominios a evaluar a partir del plan de pruebas definido con los datos de la fase anterior.
* **Escáner completo:** A partir del escáner regular, para obtener el reporte definitivo después de revisar los errores encontrados para descartar los no relevantes y los falsos positivos.

***Generación de informes.***

Como resultado el proceso de ejecución de las pruebas de seguridad generaremos los siguientes documentos.

* Definición del plan de pruebas de seguridad.
* Reporte análisis estático de código.
* Plan pruebas para el análisis dinámico.
* Reporte análisis dinámico
* Informe resultado ejecución pruebas de seguridad
  + 1. Alcance y términos de la prueba de intrusión.

Para cubrir este punto se genera un documento de plan de pruebas para cada sistema objetivo, este documento es importante para fijar el objetivo sobre el que se ejecutará la prueba de penetración, los tipos de ataque que se ejecutaran, herramientas y la metodología que se utilizaran, así como los documentos que se entregaran como resultado de la ejecución del proceso de pentesting.

Normalmente este documento se suele entregar al dueño del sistema objetivo de la prueba de penetración para que lo valide y de su aceptación antes de iniciar el proceso de pentesting.

También, en muchos casos es necesario, sobre todo en sistemas de producción, restricciones horarias a la hora de ejecutar el proceso.

Este documento, en caso de legar a fases de explotación y postexplotación, debe ser ampliado para detallar en que consistirán el proceso de explotación y postexplotación y los posibles efectos que pueden acarear al sistema de pruebas.

Para los sistemas de prueba elegidos se han creado dichos los documentos en la siguiente [ruta](https://github.com/M0l1n3ta/PFG/tree/master/Reportes/PPR%20-%20Plan%20de%20pruebas) (<https://github.com/M0l1n3ta/PFG/tree/master/Reportes/PPR%20-%20Plan%20de%20pruebas>).

* + 1. Recolección de información.

Para este punto de proceso se suelen utilizar numerosas herramientas y depende mucho de la persona que ejecute el proceso de pentesting.

Pero siempre es de utilidad partir de un análisis estático de código ya que el resultado de dicho análisis ayuda enormemente al pentester de distintas formas:

* Las vulnerabilidades encontradas en dicho proceso pueden ser utilizadas para utilizar las herramientas más oportunas para cada error detectado, así como crear las pruebas necesarias para reproducir los ataques necesarios para explotar dicho error.
* Las vulnerabilidades encontradas en este proceso son añadidas como base para el análisis. dinámico
* Suele acortar los tiempos del proceso de recolección de información.

Como resultado del análisis estático de código se genera un reporte donde, a parte de enumerar los errores encontrados y las distintas métricas de calidad del código, se añade un apartado dedicado a detallar las vulnerabilidades encontradas, así como las medidas que deben ser tenidas en cuenta para mitigar o resolver el defecto encontrado.

Este reporte puede ser presentado ante el dueño del sistema de prueba y el equipo de desarrollo para abordar mejor el proceso de solución de los defectos encontrados, así como mejoras en la calidad del código más allá de los defectos de seguridad.

Además, se suele instruir al equipo de desarrollo para que hagan uso de una de las ventajas que tiene SonarQube que es la integración con los entornos de desarrollo a través de SonarLint. Esto permite a los desarrolladores estar al tanto de los defectos que presenta el software directamente en el entorno de desarrollo y solventar los defectos antes de que se realice el análisis estático de código de las versiones entregadas. El del complemento SonarLint ha permitido en multitud de proyecto la detección temprana de defectos y su solución, así como la mejora de las distintas métricas de calidad del código.

La generación de dicho reporte se ha automatizado en un pequeño programa que detallamos en el siguiente apartado.

* + 1. Generación reporte análisis estático de código

Para generar los reportes de análisis estático de código nos ayudaremos de una pequeña utilidad creada en Python, que hemos denominado **“SonarQube Reporting Tool”,** implementada en Python que hace uso de los servicios web disponibles en SonarQube para recopilar los datos de los escáneres realizados e integrarlos con una plantilla base del reporte para generar un reporte final con los datos extraídos por la herramienta más los comentarios del pentester.

La aplicación está disponible en GitHub, la podemos descargar en instalar con los siguientes comandos:

git clone <https://github.com/M0l1n3ta/SonarQubeReportingTool.git>

cd SonarQubeReportingTool

pip3 install -r requirements.txt

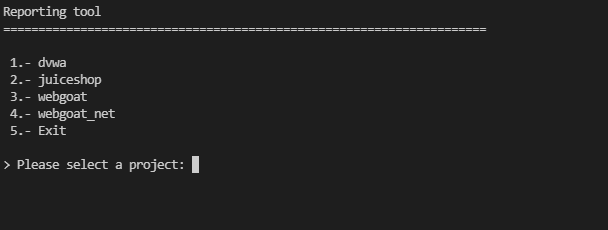
Para ejecutar la aplicación

python3 static\_analysis\_report\_generator.py

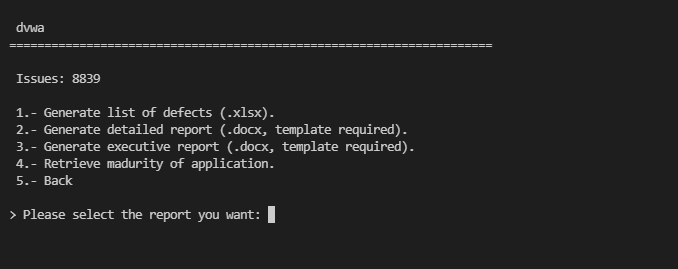
Para generar el reporte seleccionamos la primera opción “Reporting tool”:



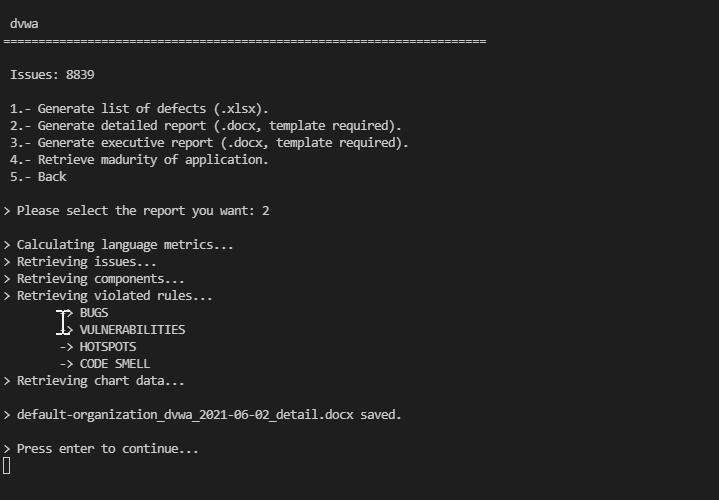
Seleccionamos el Proyecto de Sonarqube del cual queremos generar el reporte estático de código:



Finalmente seleccionamos la segunda opción **“Generate detailed report”** para generar el reporte:

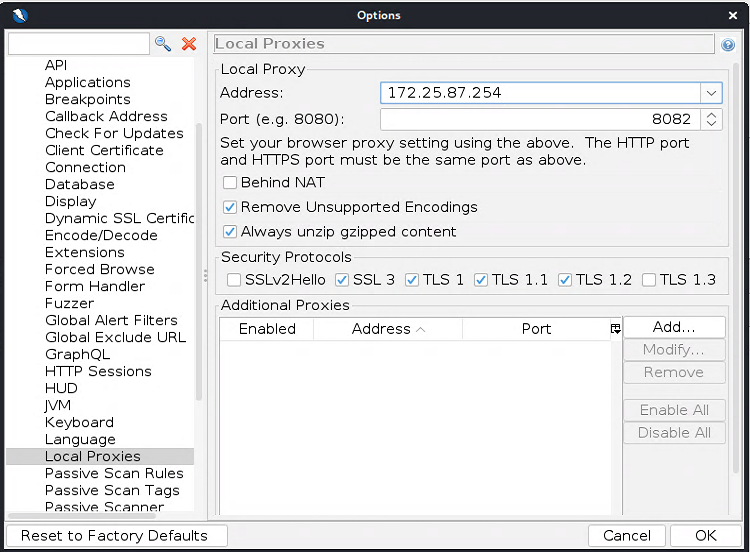


El reporte se generará en el directorio de ejecución:



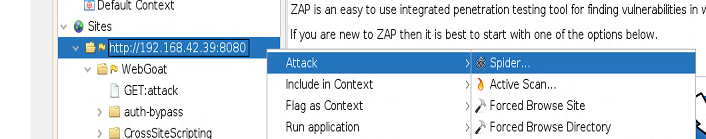
* + 1. Análisis de vulnerabilidades.

Con la información de recogida en la fase anterior más y la que pueda añadir el pentester se crea un plan de pruebas para su ejecución configurando las peticiones para que pasen a través del proxy de OWASP ZAP

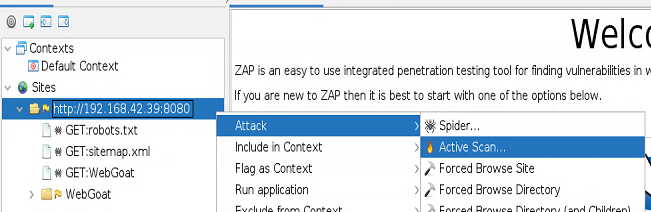


Con todas las peticiones generadas por el plan de pruebas más las que el pentester decida añadir se irán almacenado en una sesión de OWASP ZAP.

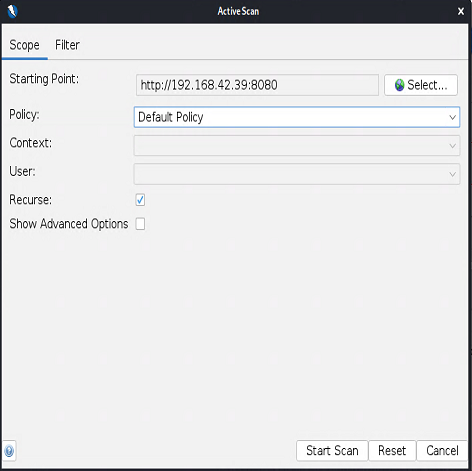
Una vez finalizado el plan de pruebas lanzamos el spider para que descubra nuevas rutas en la aplicación



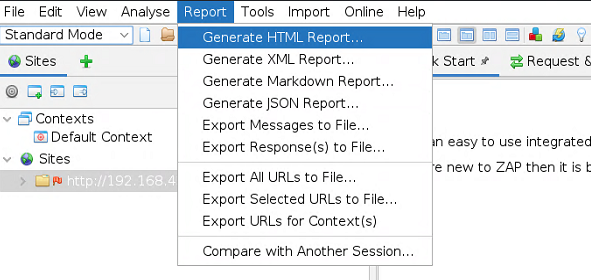
Con la url añadidas desde el plan de pruebas, más la detectadas con el proceso de “Spider”, más la que el pentester considere añadir lanzamos el proceso de análisis dinámico de código. Para ello seleccionamos la url de pruebas y con el segundo botón seleccionamos **“Attack > Active Scan”**



Para iniciar el escáner dinámico seleccionamos la política **“Default Policy”** e iniciamos con **“Start Scan”:**



Una vez finalizado escáner dinámico se revisan las alertas para descartar los que se consideren falsos positivos y se genera el reporte de análisis dinámico. Para generar el reporte de análisis dinámico se selecciona en menú la opción **“Report”**



Y seleccionamos uno de los tipos de reporte:

* HML
* XML
* Markdown
* JSON

En nuestro caso seleccionaremos el tipo de reporte HTML. Una vez generado el reporte de análisis dinámico es recomendable abrir todos los defectos encontrados en un gestor de defectos, tanto los defectos encontrados en el análisis estático de código como en el dinámico, e informar al equipo de desarrollo de los mismo y si fuese necesario planificar una reunión para abordar la solución a los defectos encontrados.

Trascurrido un tiempo se suele generar un reporte de resultado del análisis de seguridad de la versión del software analizada o antes del análisis de una nueva versión del software donde se detallará el estado de los defectos encontrados en ese momento.

* + 1. Generación de Informes

Como resultado el proceso de ejecución de las pruebas de seguridad generaremos los siguientes documentos:

* Definición del plan de pruebas de seguridad.
* Reporte análisis estático de código.
* Informe plan pruebas para el análisis dinámico.
* Reporte análisis dinámico.
* Informe resultado ejecución pruebas de seguridad
  1. Infraestructura de pruebas

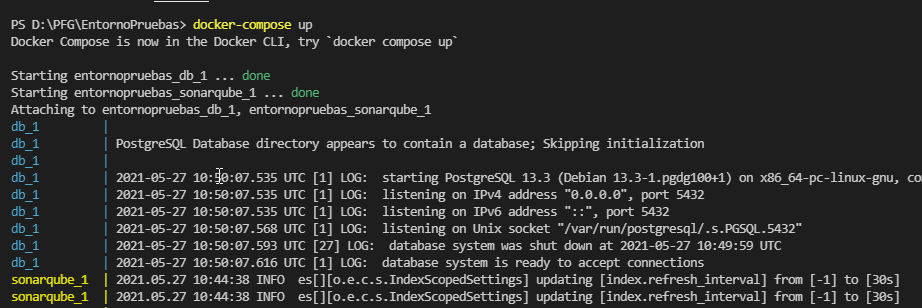
Como entorno de pruebas para la ejecución de los análisis de código; haremos uso de una máquina física y de un contenedor de Docker con la siguientes características y herramientas instaladas en cada una de ellas:

|  |  |  |
| --- | --- | --- |
| **Características** | **Máquina Física** | **Contenedor** |
| **Sistema Operativo** | Windows 10 Pro | Debian GNU/Linux 10 (buster) |
| **Herramientas** | OWASP Zap 2.10 Dependency-check  SonarScaner 4.6.2 | SonarQube 8.2  PostGresSQL 13.3 |

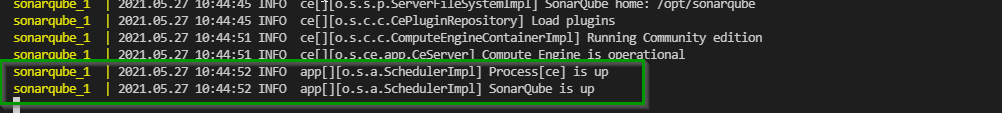
Para levantar el contenedor podemos hacer uso de docker compose incluido en la carpeta **“EntornoPrueba"** dentro de las [fuentes del proyecto](https://github.com/M0l1n3ta/PFG/tree/master)

Para levantar el entorno ejecutamos:

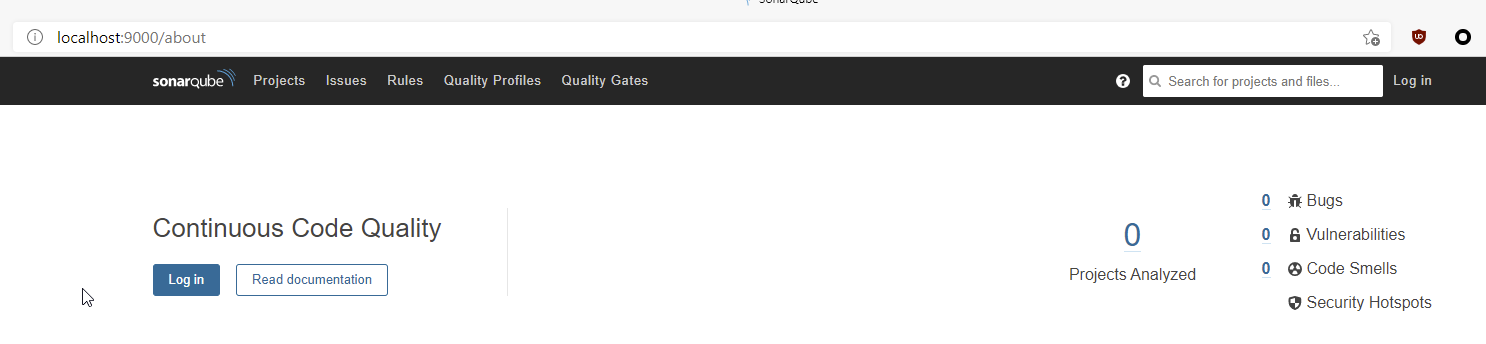
docker-compose up



Una vez que se vena las siguientes líneas en el log:



Podremos acceder a la página de SonarQube en la url <http://localhost:9000>



Docker Compose SonarQube:

version: "2"

services:

  sonarqube:

    image: molineta/sonarqube:8.2

    depends\_on:

      - db

    ports:

      - "9000:9000"

    networks:

      - sonarnet

    environment:

      SONARQUBE\_JDBC\_URL: jdbc:postgresql://db:5432/sonar

      SONARQUBE\_JDBC\_USERNAME: sonar

      SONARQUBE\_JDBC\_PASSWORD: sonar

    volumes:

      - ./extension/plugins:/opt/sonarqube/extensions/plugins

      - ./extension/sonarqube\_conf:/opt/sonarqube/conf

  db:

    image: postgres

    networks:

      - sonarnet

    environment:

      POSTGRES\_USER: sonar

      POSTGRES\_PASSWORD: sonar

    volumes:

      - postgresql:/var/lib/postgresql

      - postgresql\_data:/var/lib/postgresql/data

networks:

  sonarnet:

    driver: bridge

volumes:

  sonarqube\_data:

  sonarqube\_extensions:

1. Ejecución casos de prueba
   1. Aplicación en desarrollo de aplicaciones Web

Como aplicaciones para los casos de prueba hemos elegido una aplicación para cada una de las tecnologías más representativas en el desarrollo web a largo de estos años:

* PHP
* JavaScript
* Java
* .Net

Las aplicaciones que analizaremos serán las siguientes:

|  |  |
| --- | --- |
| **Aplicación** | **Tecnologías utilizadas** |
| [**WebGoat**](https://github.com/WebGoat/WebGoat) | Java, Spring Boot |
| [**Damn Vulnerable Web application (dwva)**](https://dvwa.co.uk/) | PHP |
| [**Juice Shop**](https://github.com/bkimminich/juice-shop) | JavaScript, Angular, Node.js |
| [**WebGoat.Net**](https://github.com/tobyash86/WebGoat.NET) | DotNet 5 |

* + 1. WebGoat

Siguiendo las tareas del documento de plan pruebas para este proyecto, realizamos las tareas que se detallan a continuación.

Para ejecutar el análisis de dependencias desde Maven, debemos añadir la siguiente configuración del plugin de Dependency-Check:

<plugin>

  <groupId>org.owasp</groupId>

  <artifactId>dependency-check-maven</artifactId>

  <version>6.1.6</version>

  <executions>

    <execution>

      <goals>

        <goal>check</goal>

      </goals>

    </execution>

  </executions>

</plugin>

A parte de la configuración anterior debemos añadir las siguientes propiedades:

<properties>

    <dependency-check-maven.version>6.1.6</dependency-check-maven.version>

    <sonar.dependencyCheck.htmlReportPath>./reports/dependency-check-report.html</sonar.dependencyCheck.htmlReportPath>

    <sonar.dependencyCheck.summarize>true</sonar.dependencyCheck.summarize>

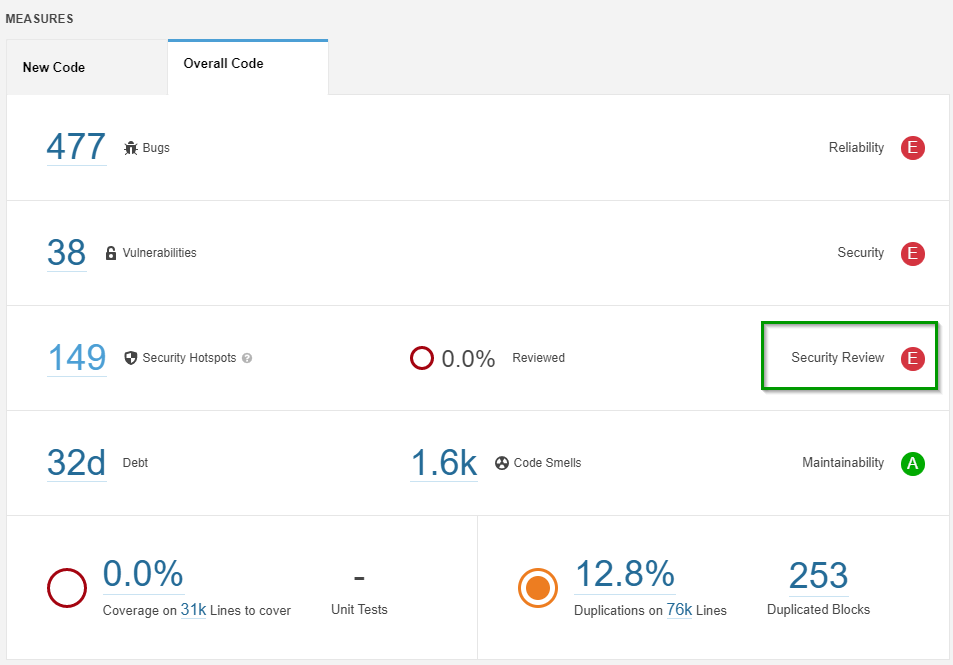
    <sonar.host.url>http://localhost:9000/</sonar.host.url>

</properties>

Para ejecutar el escáner:

mvn dependency-check:check

La ejecución del análisis estático de código, así como el análisis de dependencias, lo realizaremos a través de un [script](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/STAT/RunSonarScaner_WebGoat.ps1), con el cual obtenemos el siguiente resultado:



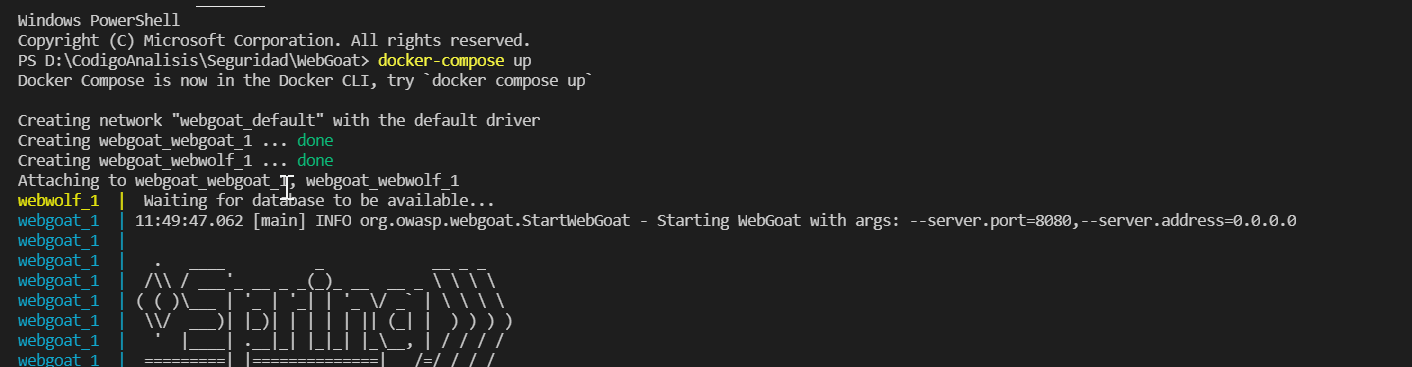
Como era de esperar obtiene el peor resultado posible en la medida de seguridad **“E”**

Todos los detalles de cada una de las vulnerabilidades, así como los **“hotspot”** están detalladas en el reporte de [análisis estático de código](https://github.com/M0l1n3ta/PFG/blob/master/Reportes/An%C3%A1lisis%20estatico%20de%20c%C3%B3digo/ReporteAnalisisestatico_WebGoat.docx).

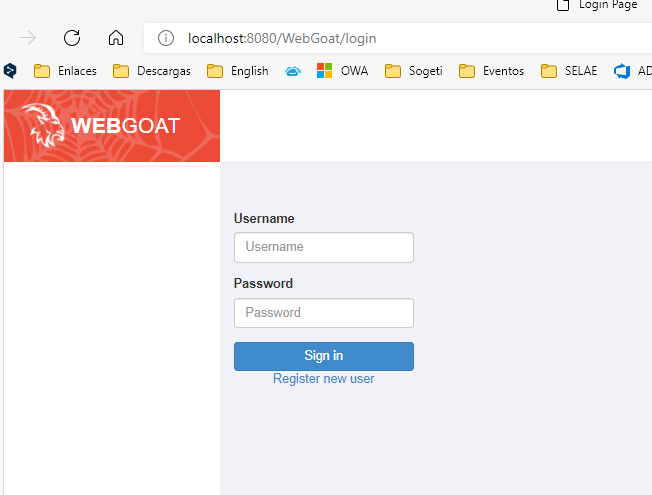
Con los defectos encontrados con el análisis estático más la información que el pentester recolecte sobre la aplicación analizada se procede a crear un [plan de prueba](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/Plan%20Pruebas/PlanPruebas_OWASP_WebGoat.ps1) para la aplicación para poder capturar los hallazgos en la herramienta de análisis dinámico (OWASP ZAP).

Antes de ejecutar el plan de pruebas levantamos la aplicación levantando el docker compose incluido en las fuentes del proyecto, para ello ejecutamos el comando:

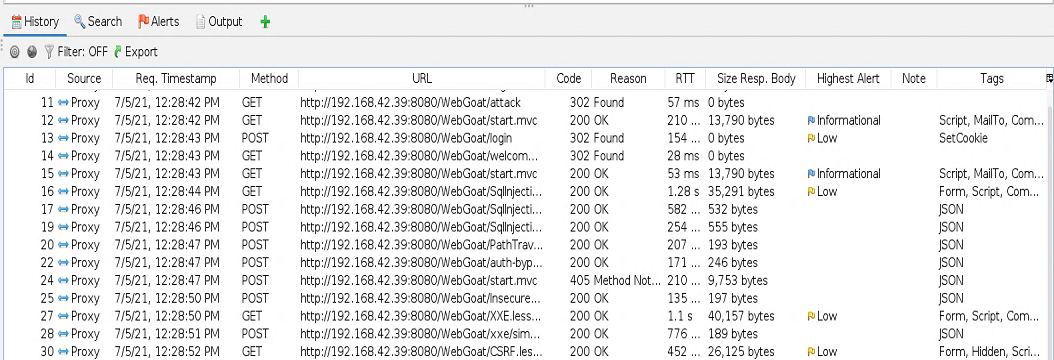
docker-compose up



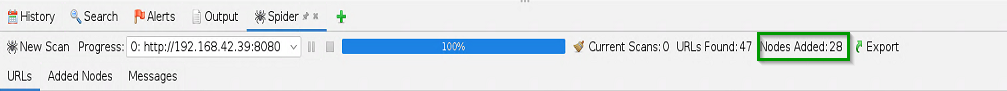
Una vez levantada la aplicación verificamos que funciona accediendo a la ruta <http://localhost:8080/WebGoat>



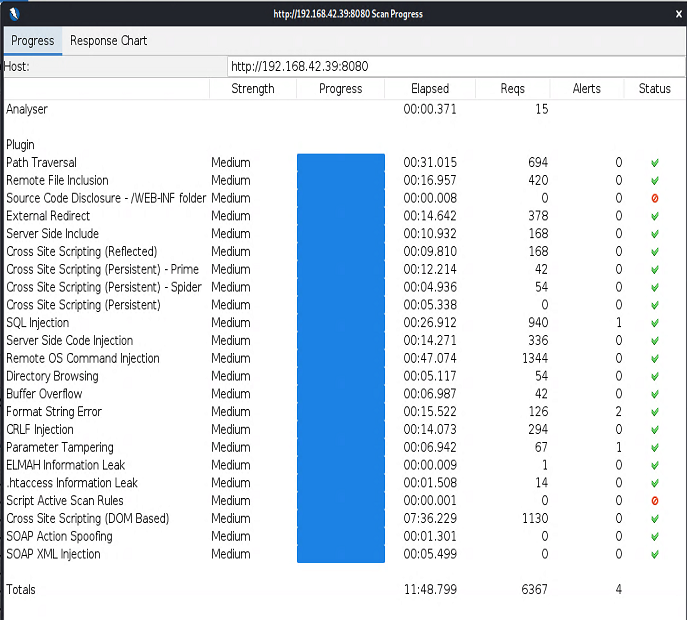
Una vez verificado, lanzamos el script de pruebas y revisamos que se vayan capturando la peticiones en OWASP ZAP correctamente



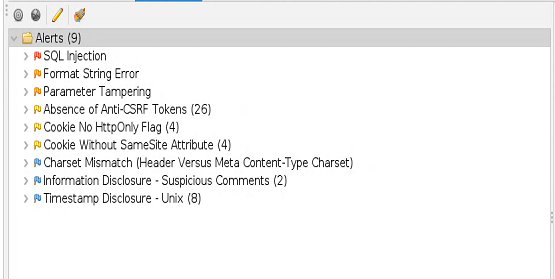
Una vez finalizado el plan de pruebas lanzamos el spider para que descubra nuevas rutas en la aplicación. Una vez finalizado el proceso de Spider veremos la nuevas url que ha detectado, en este caso 28:



Una vez finalizado el escáner en la ventana de progreso



Podremos ver las incidencias que ha detectado en análisis dinámico de la aplicación:

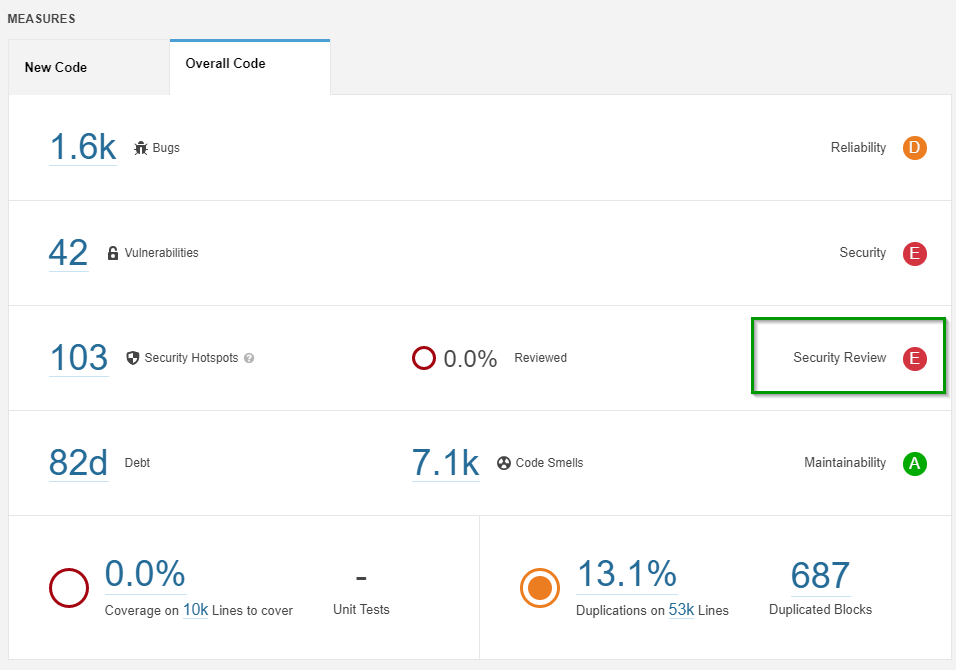


* + 1. Damn Vulnerable Web application

Siguiendo las tareas del documento de plan [pruebas para este proyecto](https://github.com/M0l1n3ta/PFG/blob/master/Reportes/DVWA/PPR%20DVWA%20-%20Plan%20Pruebas%20de%20Seguridad.docx), realizamos las tareas que se detallan a continuación.

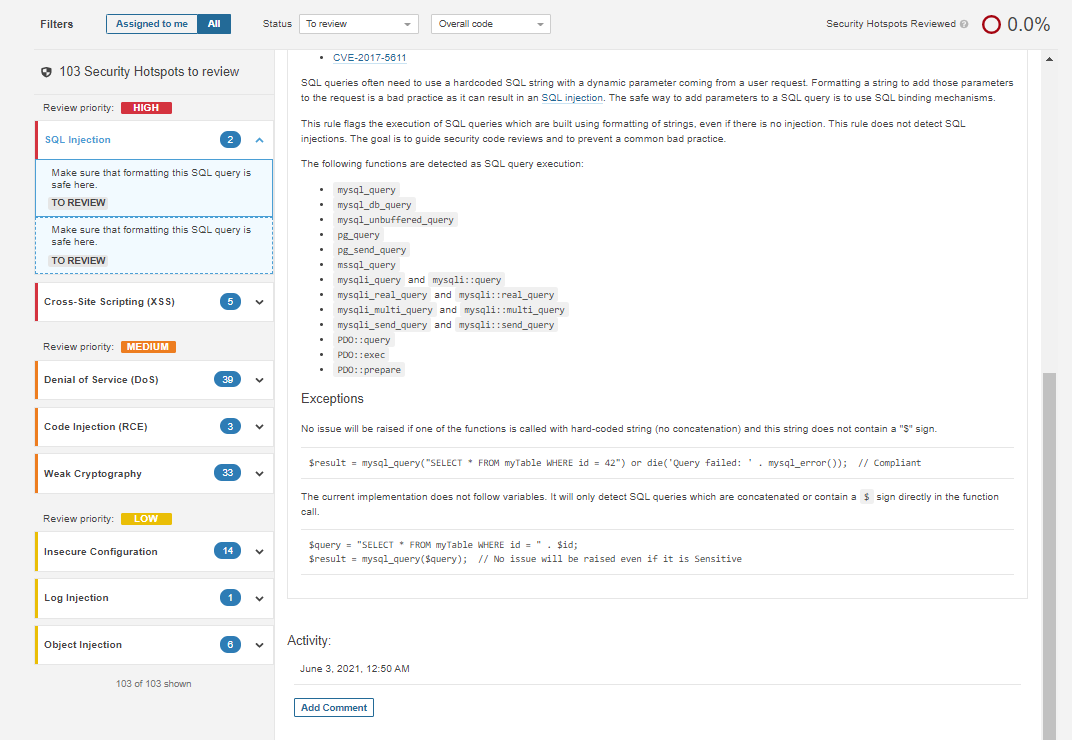
Para este proyecto no se realizará análisis de dependencias puesto que el proyecto no hace uso del componente de PHP necesario para realizar este tipo de análisis en proyectos PHP ([Composer](https://getcomposer.org/))

La ejecución del análisis estático de código lo realizaremos a través de un [script](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/STAT/RunSonarScaner_DWVA.ps1), con el cual obtenemos el siguiente resultado:



Como era de esperar obtiene el peor resultado posible en la medida de seguridad **“E”**

En el apartado de ***“Security hot spots***” vemos que se detectan la mayoría de las vulnerabilidades recogidas en el OWASP top 10:



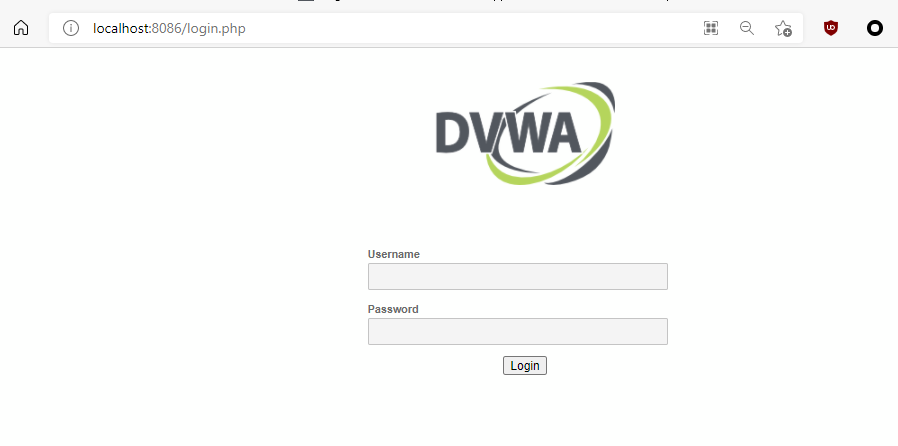
Todos los detalles de cada una de las vulnerabilidades, así como los **“hotspot”** están detalladas en el reporte de [análisis estático de código](https://github.com/M0l1n3ta/PFG/blob/master/Reportes/An%C3%A1lisis%20estatico%20de%20c%C3%B3digo/ReporteAnalisisestatico_dvwa.docx).

Con los defectos encontrados con el análisis estático más la información que el pentester recolecte sobre la aplicación analizada se procede a crear un [plan de prueba](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/Plan%20Pruebas/PlanPruebas_DVWA.ps1) para la aplicación para poder capturar los hallazgos en la herramienta de análisis dinámico (OWASP ZAP).

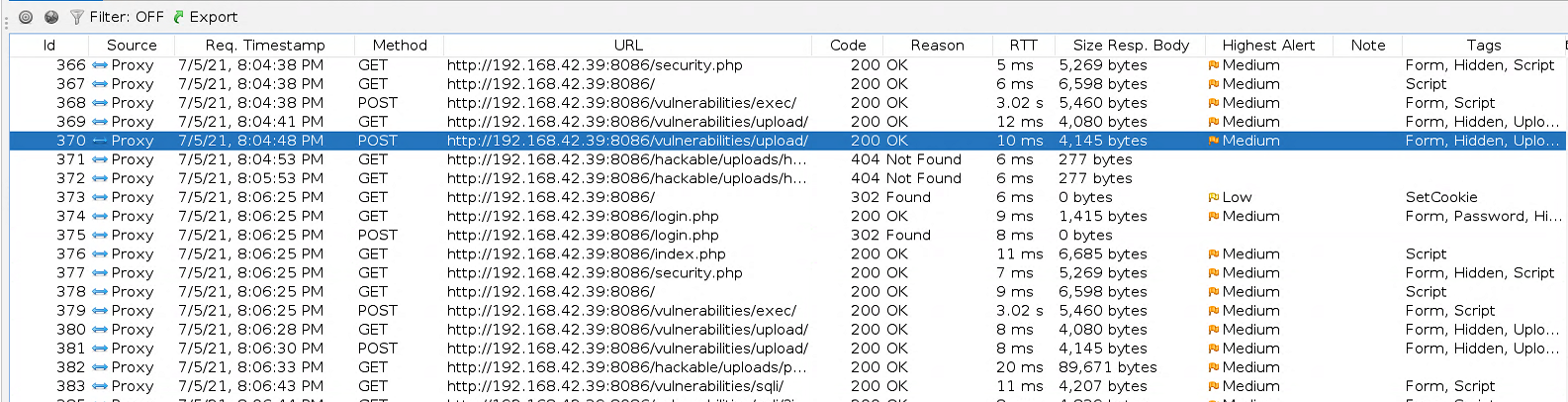
Antes de ejecutar el plan de pruebas levantamos la aplicación levantando compilando el [contenedor](https://github.com/M0l1n3ta/PFG/blob/master/EntornoPruebas/dvwa/dockerfile) incluido en las de este PFG, puesto que el proyecto original no lo incluye, o ejecutándolo compilado desde Docker Hub con el siguiente comando:

docker run --rm -it -p 8086:80 molineta/dvwa:2.0.1

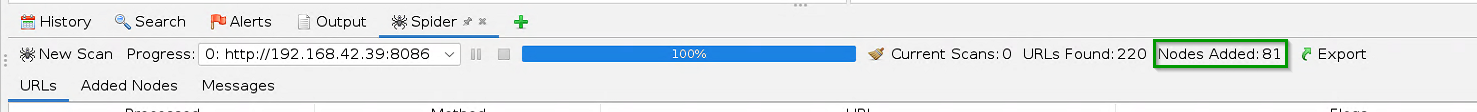
Una vez ejecutado verificamos que el contenedor se encuentra funcionando, accediendo a la siguiente ruta <http://localhost:8086/login.php>



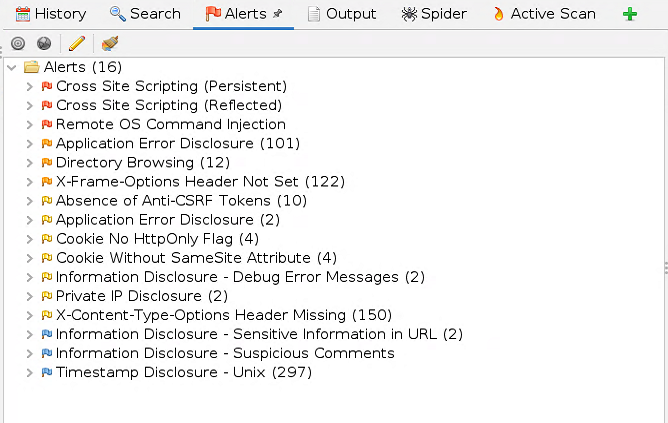
Una vez verificado, lanzamos el script de pruebas y revisamos que se vayan capturando la peticiones en OWASP ZAP correctamente



Una vez finalizado el plan de pruebas lanzamos el spider para que descubra nuevas rutas en la aplicación. Una vez finalizado el proceso de Spider veremos la nuevas url que ha detectado, en este caso 81:



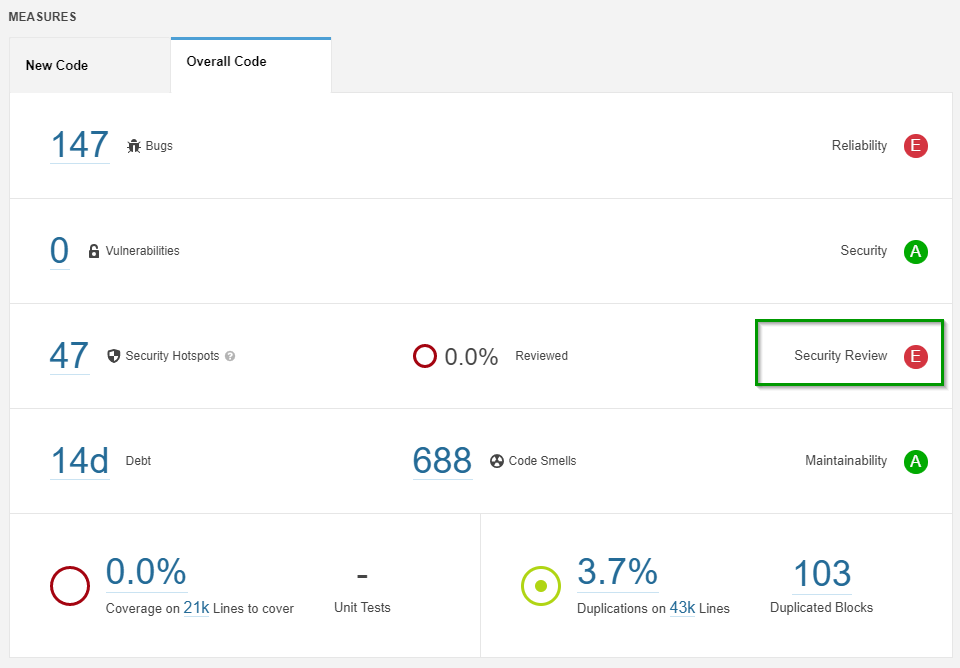
Con la url añadidas desde el plan de pruebas, más la detectadas con el proceso de “Spider”, más la que el pentester considere añadir lanzamos el proceso de análisis dinámico de código con la política ***“Default Policy”*** y esperamos que termine para ver las incidencias que ha detectado en análisis dinámico de la aplicación:



* + 1. Juice Shop

Siguiendo las tareas del documento de plan pruebas para este proyecto, realizamos las tareas que se detallan a continuación.

La ejecución del análisis estático de código, así como el análisis de dependencias, lo realizaremos a través de un [script](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/STAT/RunSonarScaner_JuiceShop.ps1), con el cual obtenemos el siguiente resultado:



Como era de esperar obtiene el peor resultado posible en la medida de seguridad **“E”**

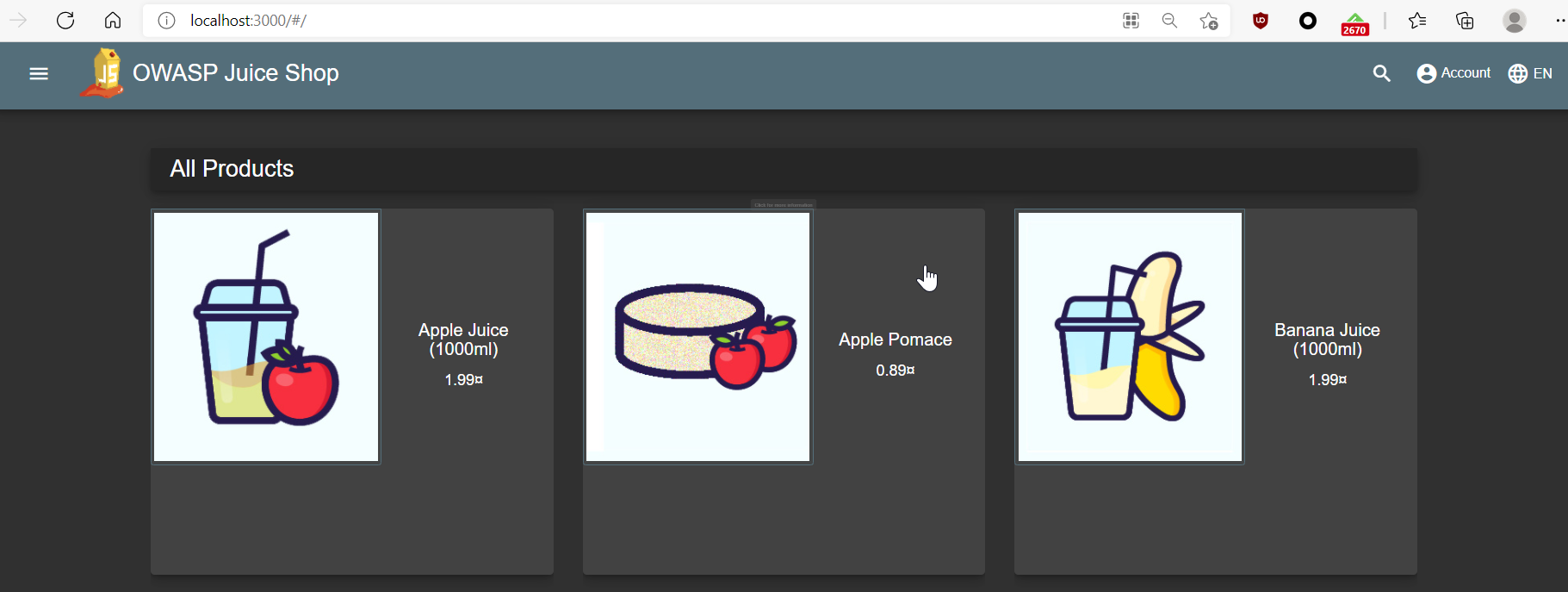
Todos los detalles de cada una de las vulnerabilidades, así como los **“hotspot”** están detalladas en el reporte de [análisis estático de código](https://github.com/M0l1n3ta/PFG/blob/master/Reportes/An%C3%A1lisis%20estatico%20de%20c%C3%B3digo/ReporteAnalisisestatico_JuiceShop.docx).

Con los defectos encontrados con el análisis estático más la información que el pentester recolecte sobre la aplicación analizada se procede a crear un [plan de prueba](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/Plan%20Pruebas/PlanPruebas_OWASP_JuiceShop.ps1) para la aplicación para poder capturar los hallazgos en la herramienta de análisis dinámico (OWASP ZAP).

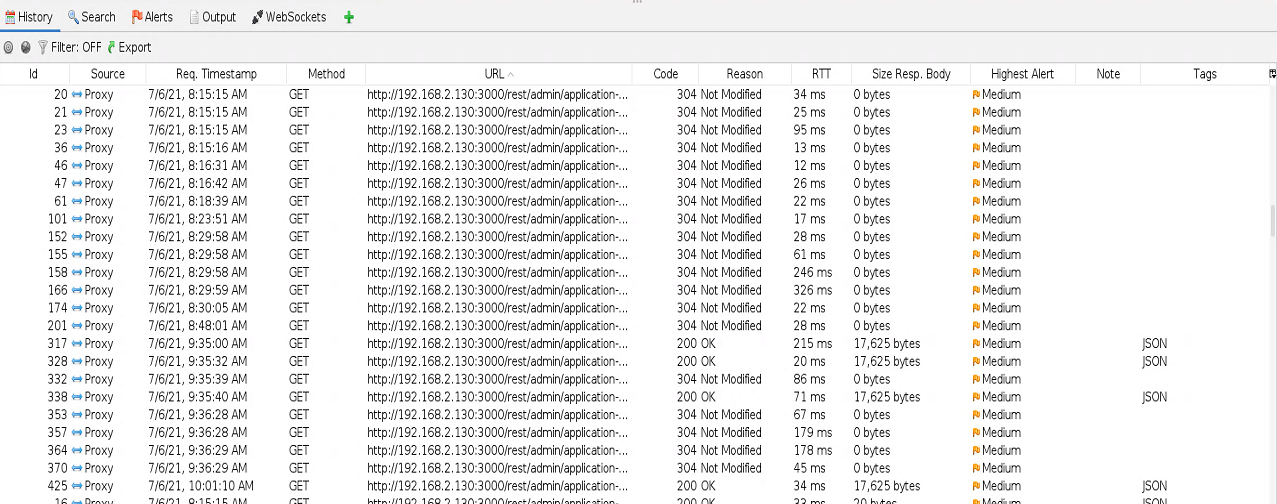
Antes de ejecutar el plan de pruebas levantamos la aplicación compilando el contenedor incluido en las fuentes del proyecto o ejecutándolo compilado desde Docker Hub con el siguiente comando:

docker run --rm -it -p 3000:3000 molineta/juiceshop:12.7.2

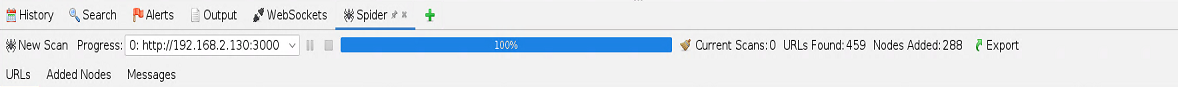
Una vez ejecutado verificamos que el contenedor se encuentra funcionando, accediendo a la siguiente ruta <http://localhost:3000>



Una vez verificado, lanzamos el script de pruebas y revisamos que se vayan capturando la peticiones en OWASP ZAP correctamente



Una vez finalizado el plan de pruebas lanzamos el spider para que descubra nuevas rutas en la aplicación. Una vez finalizado el proceso de Spider veremos la nuevas url que ha detectado, en este caso 288:



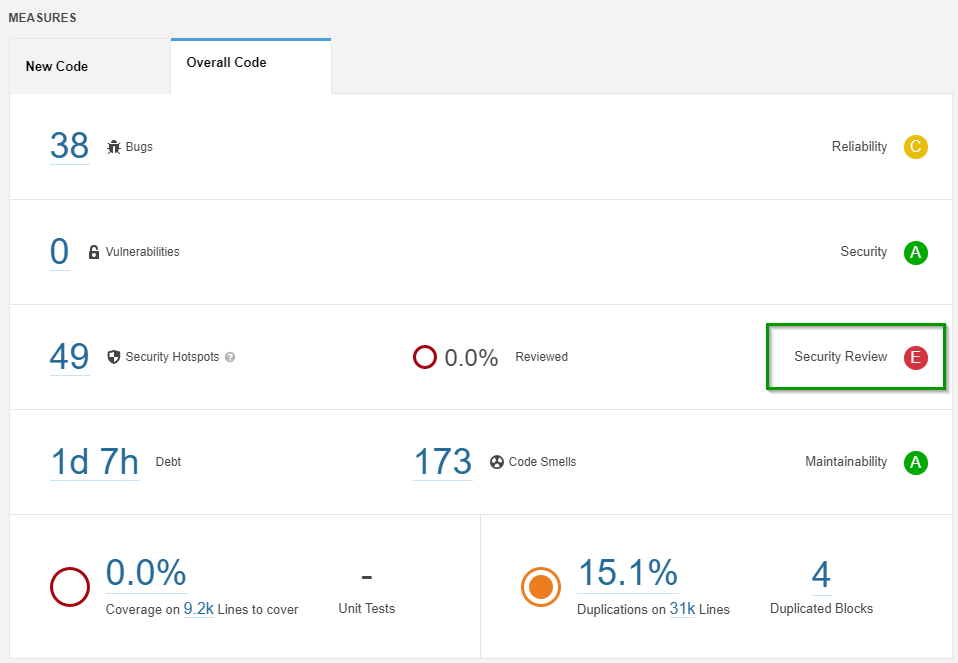
Con la url añadidas desde el plan de pruebas, más la detectadas con el proceso de “Spider”, más la que el pentester considere añadir lanzamos el proceso de análisis dinámico de código con la política ***“Default Policy”*** y esperamos que termine para ver las incidencias que ha detectado en análisis dinámico de la aplicación:



* + 1. WebGoat.Net

Siguiendo las tareas del documento de plan pruebas para este proyecto, realizamos las tareas que se detallan a continuación.

La ejecución del análisis estático de código, así como el análisis de dependencias, lo realizaremos a través de un [script](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/STAT/RunSonarScaner_WebGoat.NET.ps1), con el cual obtenemos el siguiente resultado:



Como era de esperar obtiene el peor resultado posible en la medida de seguridad **“E”**

Todos los detalles de cada una de las vulnerabilidades, así como los **“hotspot”** están detalladas en el reporte de [análisis estático de código](https://github.com/M0l1n3ta/PFG/blob/master/Reportes/An%C3%A1lisis%20estatico%20de%20c%C3%B3digo/ReporteAnalisisestatico_WebGoat_Net.docx).

Con los defectos encontrados con el análisis estático más la información que el pentester recolecte sobre la aplicación analizada se procede a crear un [plan de prueba](https://github.com/M0l1n3ta/PFG/blob/master/Scripts/Plan%20Pruebas/PlanPruebas_OWASP_WebGoat.Net.ps1) para la aplicación para poder capturar los hallazgos en la herramienta de análisis dinámico (OWASP ZAP).

Antes de ejecutar el plan de pruebas levantamos la aplicación compilando el contenedor incluido en las fuentes del proyecto o ejecutándolo compilado desde Docker Hub con el siguiente comando:

docker run --rm -it -p 5000:80 molineta/webgoat.net

Una vez ejecutado verificamos que el contenedor se encuentra funcionando, accediendo a la siguiente ruta [http://localhost:5000](http://localhost:5000/)

Una vez verificado, lanzamos el script de pruebas y revisamos que se vayan capturando la peticiones en OWASP ZAP correctamente

Una vez finalizado el plan de pruebas lanzamos el spider para que descubra nuevas rutas en la aplicación. Una vez finalizado el proceso de Spider veremos la nuevas url que ha detectado, en este caso 288:

Con la url añadidas desde el plan de pruebas, más la detectadas con el proceso de “Spider”, más la que el pentester considere añadir lanzamos el proceso de análisis dinámico de código con la política ***“Default Policy”*** y esperamos que termine para ver las incidencias que ha detectado en análisis dinámico de la aplicación:

1. Conclusiones y trabajo futuro
   * 1. Conclusiones

Una vez finalizado el desarrollo del trabajo creo que se ha conseguido detallar un proceso de pentesting que puede ser utilizado por cualquiera para obtener los defectos de seguridad en un proceso reproducible.

Se han detallado herramientas que pueden ser utilizadas, así como los documentos a generar de cara a una comunicación con el equipo de desarrollo enfocada a la resolución de los defectos encontrados en el software analizado.

Se ha desarrollado una utilidad que permite la extracción de la información de los análisis estáticos de código para incluirlos en reportes basados en una plantilla predefinida.

Se ha aportado un entorno de pruebas reproducible para la ejecución de análisis de código, tanto estático como dinámicos.

Se ha aplicado dicho proceso sobre distintas tecnologías generando los documentos requeridos por el proceso de pentesting para la obtención de los defectos presentes en dichas aplicaciones. Además, se han analizado dichos defectos incluyendo en los distintos documentos los planes para remediar dicho defectos para cada aplicación.

En mi opinión creo que se ha cumplido el objetivo general que tenía en mente que era detallar todo el proceso de pentesting para poder ejecutarse por parte de un pentester de forma que proporcione una base y unos documentos específicos que pueden servir de partida para la detección de los defectos de seguridad, o defectos en general presentes en cualquier código. Como se ha visto en los casos de prueba, dependerá en gran medida del conocimiento sobre la aplicación y el diseño de un plan de pruebas en constante evolución que puedan llegar a detectar el mayor número de defectos posibles presentes en un determinado código.

* + 1. Trabajo futuro

Uno paso lógico, para continuar en la línea del proyecto, sería integrar en proceso descrito en el presente PFG en un pipeline que permitiese la ejecución del proceso de forma automática en los procesos de desarrollo e integración continuos.

Algunas de las tecnologías donde se podría crear dicha pipeline podrían ser:

* Azure Devops
* Jenkins
* GitHub Actions

Para lograr integrar el proceso dentro de una pipeline habría que modificar la herramienta de creación de reportes para que aceptase parámetros en línea de comandos para permitir la integración de la herramienta en dicho proceso.

Uno de los trabajos a mejorar, del presente PFG, son lo planes de prueba de forma que detecten un mayor número de los defectos presentes en cada una de las aplicaciones de prueba.

Añadir más tecnologías de desarrollo web como puede ser el caso de .Net Core, que si bien en un principio opté por incluir tuve que descartar puesto que no encontré aplicaciones que tuviesen implementados todos los defectos del OWASP top 10 de forma similar a la aplicaciones escogidas para este PFG.

1. Anexos
   1. Glosario de términos

|  |  |
| --- | --- |
| Término | Definición |
| ASN | Un sistema autónomo (en inglés, Autónomas System: AS) se define como *“un grupo de redes*[*IP*](https://es.wikipedia.org/wiki/Protocolo_IP)*que poseen una política de rutas propia e independiente”.* Esta definición hace referencia a la característica fundamental de un Sistema Autónomo; realiza su propia gestión del tráfico que fluye entre él y los restantes Sistemas Autónomos que forman Internet. Un número de AS o ASN se asigna a cada AS, el que lo identifica de manera única a sus redes dentro de [Internet](https://es.wikipedia.org/wiki/Internet). |
| broadcast | En [Informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica), la difusión amplia, difusión ancha o ***broadcast***, es una forma de transmisión de [información](https://es.wikipedia.org/wiki/Informaci%C3%B3n) donde un nodo [emisor](https://es.wikipedia.org/wiki/Emisor) envía información a una multitud de nodos [receptores](https://es.wikipedia.org/wiki/Comunicaci%C3%B3n) de manera simultánea, sin necesidad de reproducir la misma transmisión [nodo](https://es.wikipedia.org/wiki/Nodo_(inform%C3%A1tica)) por nodo. |
| CVE | Del inglés ***“Common Vulnerabilities and Exposure”*** (CVE), es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación ***CVE-ID***.Está definido y es mantenido por [The MITRE Corporation](https://es.wikipedia.org/wiki/The_MITRE_Corporation) con fondos de la ***“***[***National Cyber Security Division***](https://es.wikipedia.org/w/index.php?title=National_Cyber_Security_Division&action=edit&redlink=1)***”*** del gobierno de los Estados Unidos de América. |
| DNS | El término DNS, del inglés ***Domain Name Service***, se refiere tanto al servicio de Nombres de Dominio, como al servidor que ofrece dicho servicio. |
| DAST | Del inglés "Dynamic Application Security Testing", término que hace referencia a las pruebas de análisis dinámicas de código. |
| exploit | Término ingles que hace referencia a una secuencia de comandos utilizados para, aprovechándose de un fallo o vulnerabilidad en un sistema, provocar un comportamiento no deseado o imprevisto. |
| fingerprinting | Término inglés que hace referencia a un conjunto de información que se puede utilizar para identificar protocolos de red, sistemas operativos, dispositivos de hardware, software, entre otras cosas. |
| firewall | Término ingles que hace referencia a cortafuegos que en informática es un sistemacuya funcionalidad básica es asegurar que todas las comunicaciones entre la red interna de una organización e Internet se realicen conforme a las políticas de seguridad de la organización. |
| ICMP | El protocolo de control de mensajes de Internet (en inglés: ***Internet Control Message Protocol*** y conocido por sus siglas ***ICMP***) es parte del conjunto de protocolos [IP](https://es.wikipedia.org/wiki/Protocolo_de_internet). Es utilizado para enviar mensajes de error e información operativa indicando, por ejemplo, que un sistema no puede ser localizado o que un servicio que se ha solicitado no está disponible. Estos mensajes del protocolo ***ICMP*** se envían a la dirección IP de origen del paquete. |
| OSINT | **Inteligencia de fuentes abierta** (en inglés **Open-source intelligence** **OSINT**) son datos recogidos de fuentes disponibles de forma pública para ser utilizados en un contexto de inteligencia. En la [comunidad de inteligencia](https://es.wikipedia.org/wiki/Servicio_de_inteligencia), el término "abiertas" se refiere a fuentes disponibles públicamente. |
| pentest | Término inglés que hace referencia las pruebas de intrusión y que consiste en evaluar los niveles de seguridad de una aplicación o sistema informático, en un entorno controlado |
| SCA | Del inglés "Static Code Analysys", término que hace referencia a las pruebas de análisis estático de código |
| SAST | Del inglés "Static Application Security Testing", término que hace referencia a las pruebas de análisis estático de código. |
| script | Término ingles que en informática hace un programa relativamente simple. |
| SSDLC | Del ingles "Secure Software Development Life Cycle". El Software Development Life Cycle (SDLC) es un proceso de desarrollo estructurado enfocado en la producción de software de calidad, con el menor costo y el periodo más corto posible de tiempo.  Un proceso seguro de SDLC, además añade procesos adicionales, encaminados a mejorar la calidad de software, tales como pruebas de penetración, revisiones de código o análisis de dependencias |
| SUT | Del ingles "Sytem Under Test", término que hace referencia a la aplicación o sistema sobre el cual se ejecutarán la pruebas. |
| TLS/SSL | Seguridad de la capa de transporte (en inglés: ***Transport Layer Security o TLS***) y su antecesor ***Secure Sockets Layer*** (***SSL***; en español capa de puertos seguros) son [protocolos criptográficos](https://es.wikipedia.org/wiki/Protocolo_criptogr%C3%A1fico) |
| vulnerability | Término inglés que hace referencia a problemas relacionados con la seguridad de la aplicación y que pueden suponer un riesgo para la integridad de la misma. |
| vulnerability assessment | Es el proceso de identificar, cuantificar y priorizar las vulnerabilidades encontradas en un sistema o aplicación. |

E

exploits · 5, 15

F

***fingerprinting*** · 8

P

pentest · 2, 4, 6

S

scripts · 8, 12

V

***vulnerabilty*** · 5

* 1. Documentos relacionados

|  |  |
| --- | --- |
| Nombre documento | Documento |
| PoliticadePruebas\_escanerRegular.xlsx |  |
| PoliticadePruebas\_escanerCompleto.xlsx |  |
| Reporte certificados TLS/SSL |  |
|  |  |

* 1. Instalación Software Necesario

|  |  |
| --- | --- |
| Software | Necesidad |
| github | Necesitamos el GitHub configurado con acceso a internet, puesto que mucho soft utilizado se descarga directamente de repositorios de código como pueden ser el ***SonarQubeTool***, OWASP Amass, testSLL, etc  Funcional configurando proxy SELAE:  git config --global http.proxy <http://172.26.132.120:3128>  Para instalar se debe ignorar o instalar el certificado de intercepción de SELAE, para ignorar el certificado:  git -c http.sslVerify=false clone <https://github.com/drwetter/testssl.sh.git>  configurar proxy en .profile  export ALL\_PROXY=<http://172.26.132.120:3128>  export HTTP\_PROXY=$ALL\_PROXY  export HTTPS\_PROXY=$ALL\_PROXY |
| nikto | Scanner Web opensource. Para su correcto funcionamiento será necesario ejecutarse con permisos de administrador(sudo)  <https://cirt.net/Nikto2> |
| nmap | Escáner de puertos para revelar servicios que no deberían estar disponibles en un entorno. Este soft debe poder ejecutarse con permisos de administrador(sudo) |
| testSSL | Soft para testear certificados y protocolos de cifrado del servidor.  Se instala directamente desde GitHub, preferiblemente en la ruta ‘/home/test/soft’  <https://github.com/drwetter/testssl.sh> |
| OWASP amass | Soft de mapeo de activos de red y recursos accesibles.  Se instala directamente desde Github, preferiblemente en la ruta ‘/home/test/soft’  <https://github.com/OWASP/Amass>  **git -c http.sslVerify=false clone** [**https://github.com/OWASP/Amass.git**](https://github.com/OWASP/Amass.git) |
| SonarQube | Software de análisis estático de código. Instalación detallada en el apartado 2 del presente documento |
| pip | curl -k [https://bootstrap.pypa.io/get-pip.py -o get-pip.py](https://bootstrap.pypa.io/get-pip.py%20-o%20get-pip.py)  python3 get-pip.py  o  sudo apt-get install python3-pip |
| maven | sudo apt-get install maven -y  configurar variables entorno:  export M2\_HOME=/usr/share/maven  export M2=$M2\_HOME/bin  export MAVEN\_OPTS=-Xms256m -Xmx512m |
| Dependency-check | git -c http.sslVerify=false clone <https://github.com/jeremylong/DependencyCheck.git> |

|  |  |
| --- | --- |
| SonarQubeTool | Soft generación de reportes de SonarQube. Este soft lo instalará Sogeti desde repositorio GitHub privado, preferiblemente en la ruta ‘/home/test/soft’  Instalación  sudo pip3 install -r /home/test/soft/SonarQubeTools/requirements.txt |

* 1. Instalación SonarQube
     1. Actualizar paquetes distro

Si es necesario actualizar los paquetes de la distro a los de la versión “Strech”

wget -q <https://www.postgresql.org/media/keys/ACCC4CF8.asc> -O - | sudo apt-key add -

sudo sh -c 'echo "deb <http://apt.postgresql.org/pub/repos/apt/> stretch-pgdg main" >> [/etc/apt/sources.list.d/pgdg.list'](https://teams.microsoft.com/_)  
 sudo sh -c 'echo "deb <http://ftp.es.debian.org/debian/> stretch main contrib non-free" >> [/etc/apt/sources.list'](https://teams.microsoft.com/_)

Una vez actualizados los paquetes actualizamos el sistema ejecutando:

apt-get upgrade && apt-get update

* + 1. Instalar y configurar PostgreSQL

Instalación de Postgre SQL

sudo apt-get install -y postgresql-12 postgresql-contrib-12 postgresql-client-12

 Una vez instalado configuramos el servicio para que se inicie con el inicio del sistema y arrancamos el servicio.

sudo systemctl enable postgresql

sudo systemctl start postgresql

Configuración PostgreSQL

sudo su - postgres

psql

CREATE USER sonar ;  
ALTER USER sonar WITH ENCRYPTED password 'p@ss1234';  
CREATE DATABASE sonardb WITH ENCODING 'UTF8' OWNER sonar ;

Salimos ejecutando el comando:

\q

 Probar conexión BBDD

psql -h localhost -U sonar -d sonardb -p 5432

* + 1. Instalación Configuración de SonarQube

Instalación SonarQube

Descargamos el soft y lo situamos en la carpeta /opt/Sonarqube con los siguientes comandos:

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.0.zip

unzip sonarqube-8.0.zip

sudo mv sonarqube-8.0 /opt/sonarqube

Configuración Sonarqube

Editamos el fichero “/opt/sonarqube/conf/sonar.properties” con las siguientes configuraciones

nano /opt/sonarqube/conf/sonar.properties

sonar.jdbc.username=sonar  
sonar.jdbc.password=p@ss1234  
sonar.jdbc.url=jdbc:postgresql://localhost:5432/sonardb

sonar.web.javaAdditionalOpts=-server

sonar.web.host=127.0.0.1  
sonar.web.port=9000

Add Sonar User and Privileges

sudo useradd sonar

echo ‘sonar:sonar’ | sudo chpasswd sonar

sudo chown -R sonar:sonar /opt/sonarqube

Ejecutar SonarQube

Modificar fichero para ejecutar sonar como usuario:

nano /opt/sonarqube/bin/linux-x86-64/sonar.sh

RUNA\_AS\_USER=sonar

 Modificar memoria de inicio:

sysctl -w vm.max\_map\_count=262144

hacer presistente en /etc/sysctl.conf

vm.max\_map\_count = 262144

Iniciar SonarQube

/opt/sonarqube/bin/linux-x86-64/[sonar.sh](http://sonar.sh/) start

Parar sonarQube

/opt/sonarqube/bin/linux-x86-64/[sonar.sh](http://sonar.sh/) stop

Configure Reverse Proxy for SonarQube(Opcional)

SonarQube listens to port 9000 by default on localhost. To access it via standard HTTP 80 port, you will need to setup reverse proxy.

En versiones recientes de SonarQube se limitó el inicio de sonar a usuarios root, esto provoca que no se pueda configurar en el fichero de propiedades del sonar valores por debajo del puerto 100. Para conseguir que el sonar responda por el puerto estándar HTTP(80) o HTTPS(443) es necesario utilizar un proxy reverso.

Para habilitar el modo proxy ejecutamos

# sudo a2enmod proxy

# sudo a2enmod proxy\_http

Proceed to set up a virtual host. Execute the following command to proceed towards creation of virtual host.

# sudo nano /etc/apache2/sites-available/sonar.conf

Add the following text in opened file, then press **Ctrl + X**, type **Y** and hit **Enter** key to save the file.

<VirtualHost \*:80>

ServerName 172.20.139.81

ServerAdmin admin@example.com

ProxyPreserveHost On

ProxyPass / http://localhost:9000/

ProxyPassReverse / http://localhost:9000/

TransferLog /var/log/apache2/sonar\_access.log

ErrorLog /var/log/apache2/sonar\_error.log

</VirtualHost>

Habilitamos el nuevo servidor virtual configurad y reiniciamos el servicio de Apache Server

# *sudo a2ensite sonar*

# *sudo systemctl restart apache2*