



Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Tecnología específica de
Ingeniería de computadores

**Implantación 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 de
Ingeniería de computadores

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

Autor: Emilio José Roldán Navarro

Tutor: Jose Luis Martinez Martinez

Junio, 2021

*A mi mujer e hijos
que aguantan mis largas horas
delante de la pantalla del ordenador.*

Declaración de autoría

Yo, Emilio José Roldán Navarro, con DNI 47069965M, declaro que soy el único autor del trabajo fin de grado titulado “Implantación de técnicas y herramientas de pentesting en el proceso de desarrollo de software”, 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 14 de Julio de 2021

Fdo.: Emilio José Roldán Navarro

Resumen

El mundo de la ciberseguridad es un mundo en constante evolución, en los últimos años ha ido evolucionando en consonancia con los cambios que se han ido realizado an el ciclo de desarrollo del software.

Actualmente los ciclos de desarrollo de sofwtware son cada vez más cortos, pero no por ello se debe descuidar la calidad del software entregado, así como asegurar que este se encuentre libre de defectos, sobre todo de defectos que involucren algún aspecto de la seguridad TI.

Es por ello, que el objetivo principal de este proyecto debe ser ayudar al pentester a realizar el trabajo que conllevan las pruebas de seguridad de una aplicación o entorno en el tiempo que marca actualmente el desarrollo y con la misma calidad o mejor.

Agradecimientos

En primer lugar, me gustaría agradecer a mi mujer por su apoyo compañía y apoyo in-cansable que me permite sacar siempre fuerzas para continuar.

A mis hijos Eloy e Isabel, que son mi mayor orgullo y alegría en este mundo.

A mi grupo de amigos de Carrera y compañeros de trabajo por aguntar mis charlas "frikis" y compartir las alegrías y penas de la vida.

A mi familia por su apoyo y compañía constantes.

Por último, me gustaría agradecer a mi tutor Jose Luis por su ayuda y apoyo para conseguir llevar este proyecto a buen puerto.

Índice general

1	Introducción	1
1.1	Motivación	2
1.2	Objetivos	2
1.3	Metodología y Competencias	3
1.4	Estructura del proyecto	3
2	Análisis del estado del arte	5
2.1	Proceso de pentesting	5
2.1.1	<i>¿Que es un prueba de penetración o pentest?</i>	5
2.1.2	<i>Fases del prueba de intrusión.</i>	5
2.2	Detalle y Clasificación de vulnerabilidades OWASP Top 10	8
2.2.1	<i>A1:2017 - Inyecciones</i>	9
2.2.2	<i>A2:2017 - Pérdida de autenticación y gestión de sesiones (Broken Authentication)</i>	11
2.2.3	<i>A3:2017 - Exposición de datos sensibles (Sensitive Data Exposure)</i>	12
2.2.4	<i>A4:2017 - XML External Entities (XXE)</i>	12
2.2.5	<i>A5:2017 - Pérdida de control de acceso (Broken Access Control)</i>	14
2.2.6	<i>A6:2017 - Configuración de seguridad incorrecta (Security Misconfiguration)</i>	14
2.2.7	<i>A7:2017 - Secuencia de comando de sitios cruzados (XSS)</i>	14
2.2.8	<i>A8:2017 - Deserialización insegura (Insecure Deserialization)</i>	16
2.2.9	<i>A9:2017 - Uso de componentes con vulnerabilidades conocidas</i>	17
2.2.10	<i>2.2.10. A10:2017 - Registro y monitoreo insuficientes</i>	17
2.3	Herramientas de análisis de código	18
2.3.1	<i>Herramientas de análisis estático de código</i>	18
2.3.2	<i>Herramientas de análisis dinámico de código</i>	23

3 Diseño solución técnica.	25
3.1 Metodología de pruebas	25
3.1.1 <i>Alcance y términos de la prueba de intrusión</i>	26
3.1.2 <i>Recolección de información</i>	26
3.1.3 <i>Generación reporte análisis estático de código</i>	27
3.1.4 <i>Análisis de vulnerabilidades.</i>	30
3.1.5 <i>Generación de informes.</i>	33
3.2 Infraestructura de pruebas	34
4 Ejecución casos de prueba	37
4.1 Aplicación en desarrollo de aplicaciones Web	37
5 Conclusões y trabajo futuro	51
5.1 Conclusiones	51
5.2 Trabajo futuro	52
A Anexo 1 - Informes generados	53
A.0.1 <i>Definición del plan de pruebas de seguridad</i>	53
A.0.2 <i>Reporte análisis estático de código</i>	53
A.0.3 <i>Plan pruebas para el análisis dinámico</i>	53
A.0.4 <i>Reporte análisis dinámico</i>	53
A.0.5 <i>Informe resultado ejecución pruebas de seguridad</i>	54
Referencia bibliográfica	55

Índice de figuras

2.1	OWASP Top 10 2017	8
2.2	Aplicación vulnerable SQLi.	9
2.3	Petición normal	9
2.4	Ejemplo Ataque SQLi.	10
2.5	Resultado ataque SQLi.	10
2.6	Token JWT.	11
2.7	Token JWT decodificado.	11
2.8	Ataque Broken Authentication.	12
2.9	Ataque XXE.	13
2.10	Aplicacion insegura XXS.	14
2.11	Ataque XXS.	15
2.12	ASVS 4.0 10.0.1 Security control.	18
2.13	Versiones Sonarqube 8.9.	20
2.14	Interfaz OWASP Zap	23
3.1	SonarQube Reporting Tool.	28
3.2	Seleccionar proyecto Sonarqube.	28
3.3	Generar reporte análisis estático	29
3.4	Resultado generación reporte análisis estático	29
3.5	Configuración	30
3.6	OWASP ZAP Spider	30
3.7	OWASP ZAP Active Scan	31
3.8	OWASP ZAP Active Scan Start	31
3.9	Reporte OWASP ZAP	32
3.10	Docker compose up	34
3.11	SonarQube server running	35
3.12	SonarQube portal	35
4.1	Resultado análisis estatico código WebGoat	39
4.2	WebGoat run docker compose	40
4.3	WebGoat run docker compose	40

4.4	WebGoat Zap Proxy Requests	41
4.5	WebGoat Spider Result	41
4.6	WebGoat Active Scan Finished	42
4.7	WebGoat Active Scan Alerts	42
4.8	Resultado análisis estatico código DVWA	43
4.9	DVWA Security hotspot	44
4.10	DVWA Application running	45
4.11	DVWA Zap Proxy Requests	45
4.12	WebGoat Spider Result	46
4.13	DVWA Active Scan Alerts	46
4.14	Resultado análisis estatico código Juice Shop	47
4.15	JuiceShop Application running	48
4.16	JuiceShop Zap Proxy Requests	48
4.17	JuiceShop Spider Result	49
4.18	JuiceShop Active Scan Alerts	49

Índice de tablas

2.1	Parámetros línea comandos dependency-check	22
3.1	Infraestructura de pruebas	34
4.1	Parámetros línea comandos dependency-check	37

Índice de listados de código

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 desarrollo, pasando del ciclo clásico de desarrollo en cascada con tiempos 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 entre 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.

1.1. 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 presentados 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.

1.2. 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 más 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 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 la 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.

1.3. Metodología y Competencias

Para la realización del PFG haremos uso de la metodología [OWASP Application Security Verification Standard \(ASVS\) 4.0 \[OWASP, 2019\]](#) 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:

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

1.4. 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 qué 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 capítulo 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.

2. Análisis del estado del arte

2.1. Proceso de pentesting

2.1.1. ¿Qué es una prueba de penetración o pentest?

Según la definición de OWASP [OWASP, 2017] 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 las 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, si 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 suel 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 ejecutasesn 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 qué 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: Submit

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=5ltegf9rqvk5u6u01kkbl0gkd4; 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=5ltegf9rqvk5u6uo1kkbl0gkd4; security=low
Upgrade-Insecure-Requests: 1
```

Figura 2.4: Ejemplo Ataque SQLi.

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

The screenshot shows a web page with the title "Vulnerability: SQL Injection". Below the title is a form with a "User ID:" input field and a "Submit" button. The input field contains the value "%' union select user,password from users#". The page displays five sets of results, each showing a different user entry. The first set is "admin" and "5f4dcc3b5aa765d61d8327deb882cf99". The second set is "gordonb" and "e99a18c428cb38d5f260853678922e03". The third set is "1337" and "8d3533d75ae2c3966d7e0d4fcc69216b". The fourth set is "pablo" and "0d107d09f5bbe40cade3de5c71e9e9b7". The fifth set is "smithy" and "5f4dcc3b5aa765d61d8327deb882cf99". All results are displayed in red text.

ID:	First name:	Surname:
%' union select user,password from users#	admin	5f4dcc3b5aa765d61d8327deb882cf99
%' union select user,password from users#	gordonb	e99a18c428cb38d5f260853678922e03
%' union select user,password from users#	1337	8d3533d75ae2c3966d7e0d4fcc69216b
%' union select user,password from users#	pablo	0d107d09f5bbe40cade3de5c71e9e9b7
%' union select user,password from users#	smithy	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.eyJpYXQiOjE2MjM3NzIyNzUsImFkbWluIjoiZmFsc2UiLCJ1c2VyIjoiVG9tIn0.0BgEpj-k6Hm1uSYk8yj7zVfdIOYP1sUGoONOCGghAwyCv_QTRGamJh8sr-mpb0t5aM0J1cHaiznGwIf_dHPWfQ
X-ASSP-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

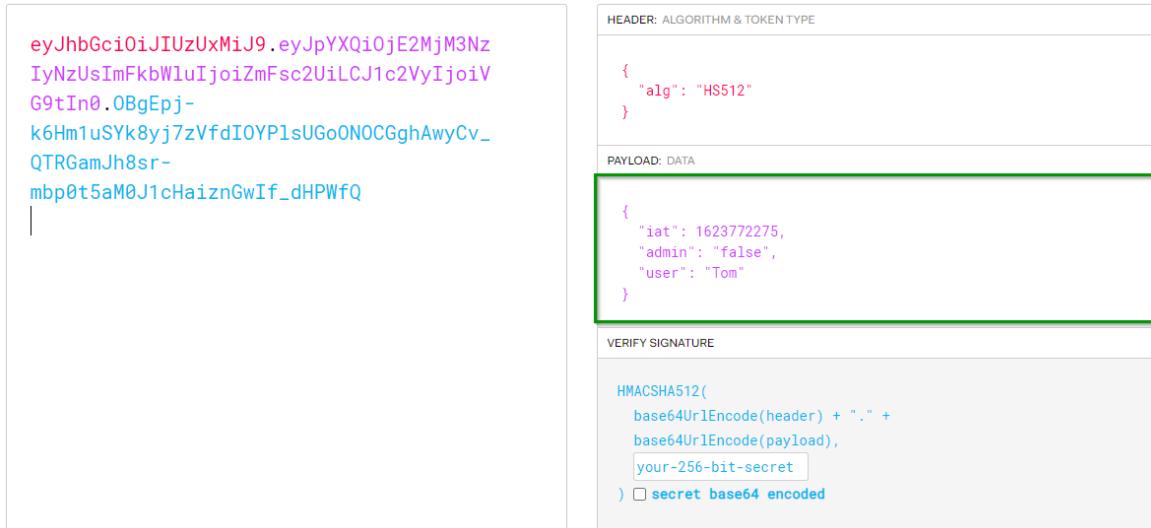


Figura 2.7: Token JWT decodificado.

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

```
POST http://192.168.43.157:8080/WebGoat/JWT/votings/Admin%20lost%20password HTTP/1.1
Host: 192.168.43.157:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: /*
Accept-Language: en-US,en;q=0.5
X-Requested-With: XMLHttpRequest
Origin: http://192.168.43.157:8080
Connection: keep-alive
Referer: http://192.168.43.157:8080/WebGoat/start.mvc
Cookie: access_token=eyJhbGciOiJIUzIwMjQyM3NzIyNzUsImFkbWluIjoiZmFsc2UlCJ1c2VyIjoiVG9tIn0.0BgEpj-k6Hm1uSYk8yj7zVfdI0YPlsUGoONOCgghAwyCv_QTRGamJh8sr-mpb05aM0J1chaznGw1f_dHPWf0; JSESSIONID=R30g91hXW8p18X9X9R22c1E50EsgKSBLoAn2XA
Content-Length: 0
```

Figura 2.8: Ataque Broken Authentication.

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

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE data [

```

Listing 1: DTD example**Figura 2.9:** Ataque XXE.

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.

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.

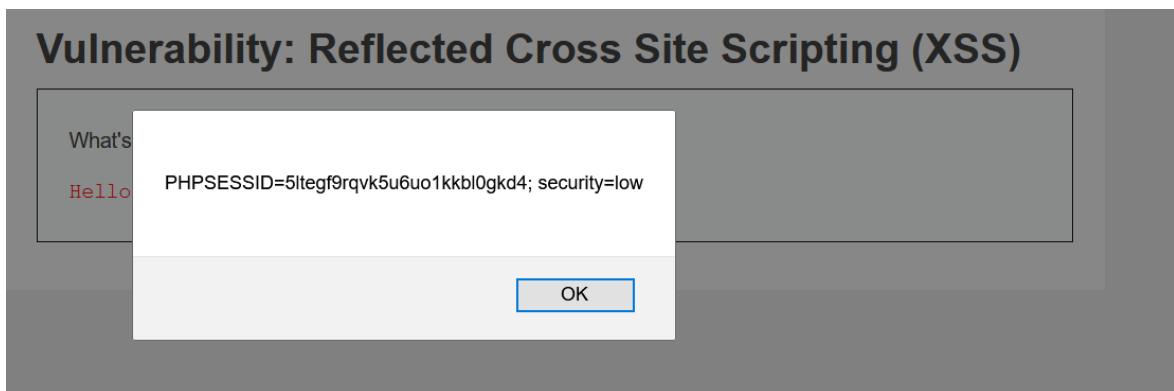


Figura 2.11: Ataque XSS.

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 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:

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

```
r00ABXNyADFvcmcuZHVtbXkuaW5zZWN1cmUuZnJhbWV3b3JrL1Z1bG51cmFibGVUYXNrSG9sZGVyAAAAAAAAAICAANMABZyZXF1ZXNOZWRFeGVjdXRpb25UaW1ldAAZTGphdmEvdG1tZS9Mb2NhxE  
RhdGVUaW1l00wACnRhctBY3RpB250ABJMamF2YS9sYW5nL1N0cmluZztMAAh0YXNrTmFtZXEaf  
gACeHBzcgANamF2YS50aW1lL1N1cpVdhLobIkiyDAAAeHB3DgUAAAf1BgYSCgML+WfAeHQAB3Ns  
ZWVwIDV0AAZteVRhc2s=
```

Dicho token al ser enviado en una petición al servidor provoca que la aplicación se detenga 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: Java Serialize Code

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

Dentro 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 hibridas 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 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.

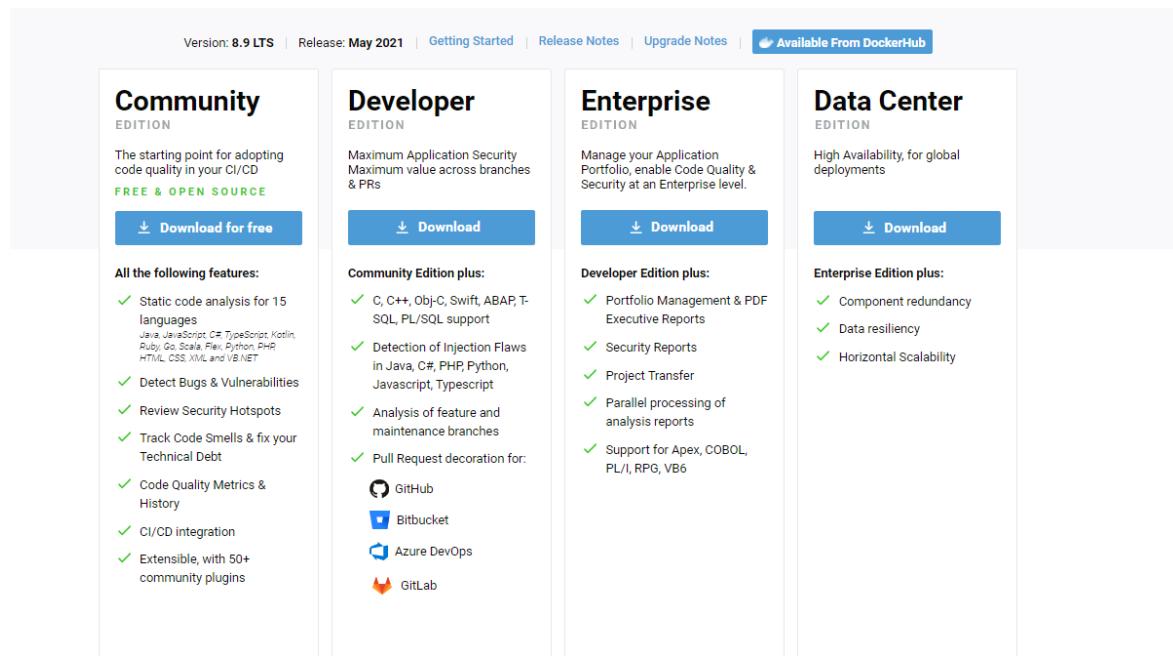


Figura 2.13: Versiones Sonarqube 8.9.

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

- Java
- JavaScript
- c#
- TypeScript
- Ruby
- 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++

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 Line Tool](#)
- Grandle plugin
- [Maven 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)

Tabla 2.1: Parámetros línea comandos dependency-check

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

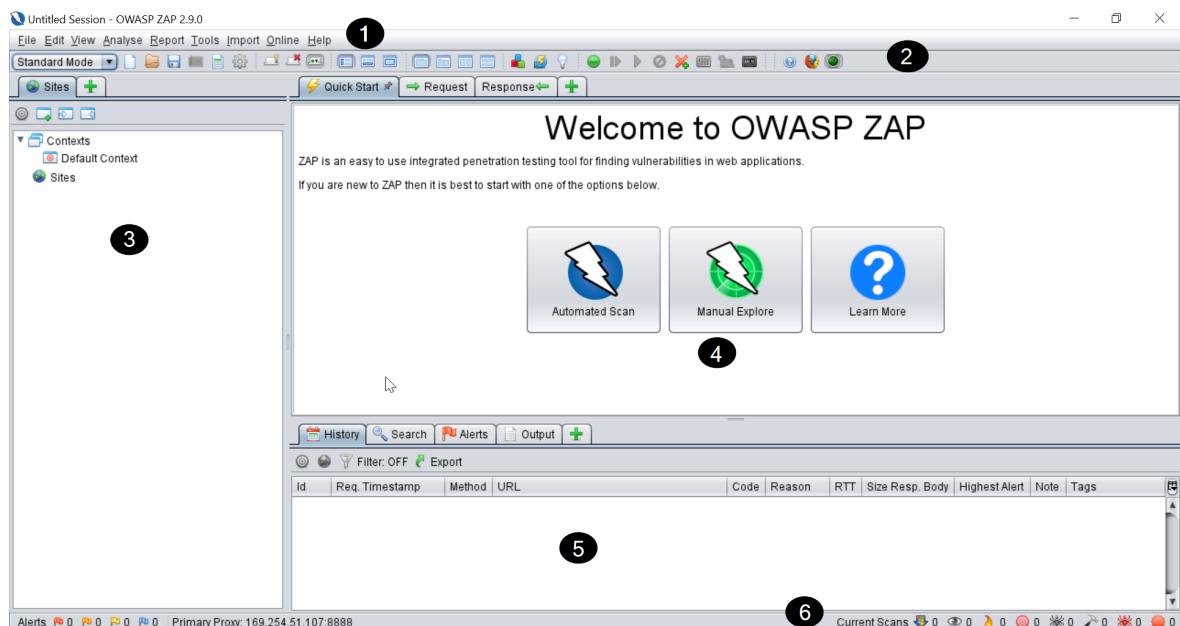


Figura 2.14: Interfaz OWASP Zap

- **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.

Fases de del proceso de pentesting aplicadas:

- Alcance y términos de la prueba de intrusión.
- Recolección de información.
- Análisis de vulnerabilidades.
- Generación de informes.

Detallaremos el proceso en las siguientes secciones.

3.1.1. 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 ejecutarán, herramientas y la metodología que se utilizarán, 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 llegar 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 acarrear al sistema de pruebas.

Para los sistemas de prueba elegidos se han creado dichos los documentos en la siguiente ruta [\[Navarro, 2021\]](#)

3.1.2. 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 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.

3.1.3. 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**”:



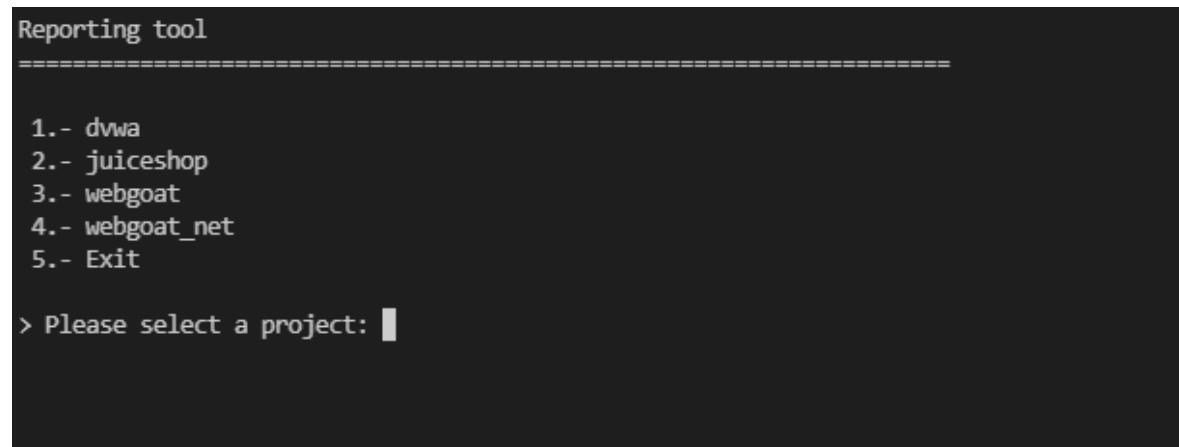
The screenshot shows a terminal window with a decorative banner at the top. Below it, a menu is displayed:

```
8888888888 .d8888b. 88888888 88888888      d8888 8888888b.
888     d88P Y88b  888  888      d88888 888  "88b
888     Y88b.    888  888      d88P888 888 .88P
88888888  "888b.  888  888      d88P 888 88888888K.
888     "Y88b.  888  888      d88P 888 888 "Y88b
888     "888  888  888      d88P 888 888 888
888     Y88b d88P 888  888      d88888888888 888 d88P
8888888888 "88888P" 88888888 88888888 d88P 888 88888888P"
```

1.- Reporting tool
2.- Utility tool
3.- Exit
> Please select an option: █

Figura 3.1: SonarQube Reporting Tool.

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



The screenshot shows a terminal window with the title “Reporting tool” and a list of projects:

```
Reporting tool
=====
1.- dwqa
2.- juiceshop
3.- webgoat
4.- webgoat_net
5.- Exit
```

> Please select a project: █

Figura 3.2: Seleccionar proyecto Sonarqube.

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

```
dwya
=====
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
```

Figura 3.3: Generar reporte análisis estático

El reporte se generará en el directorio de ejecución:

```
dwya
=====
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_dwya_2021-06-02_detail.docx saved.

> Press enter to continue...
[]
```

Figura 3.4: Resultado generación reporte análisis estático

3.1.4. 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.

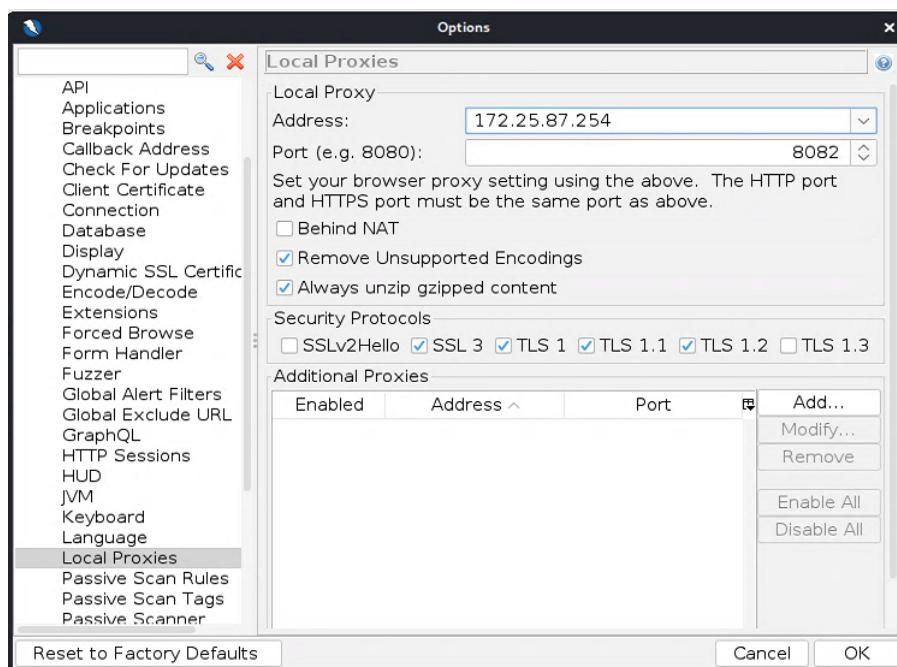


Figura 3.5: Configuración

Con todas las peticiones generadas por el plan de pruebas más las que el pentester decide 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

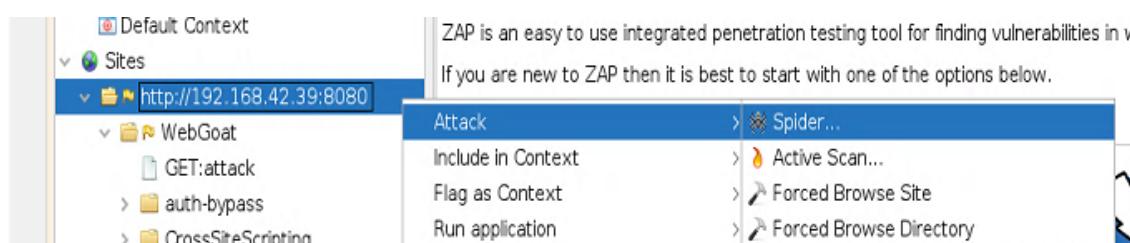


Figura 3.6: OWASP ZAP Spider

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”

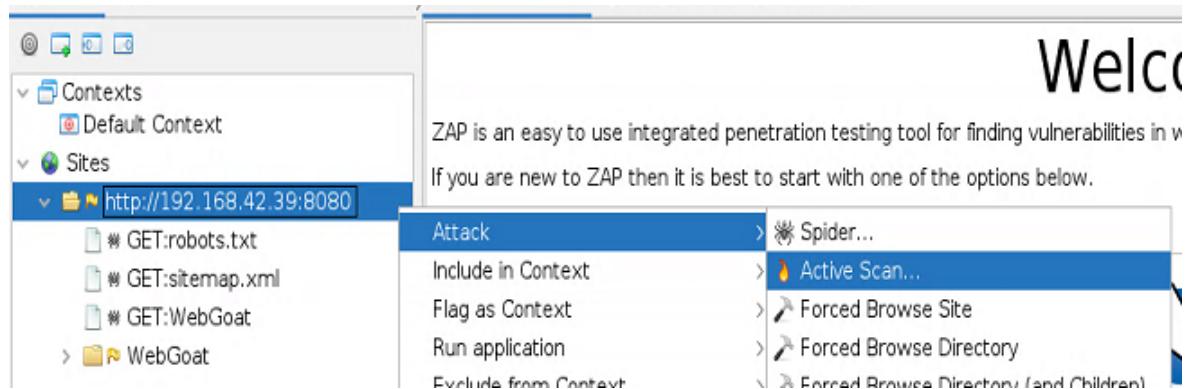


Figura 3.7: OWASP ZAP Active Scan

Para iniciar el escáner dinámico seleccionamos la política “Default Policy” e iniciamos con “Start Scan”, como se muestra en la figura /reffig:OWASPZap Active Scan Start:

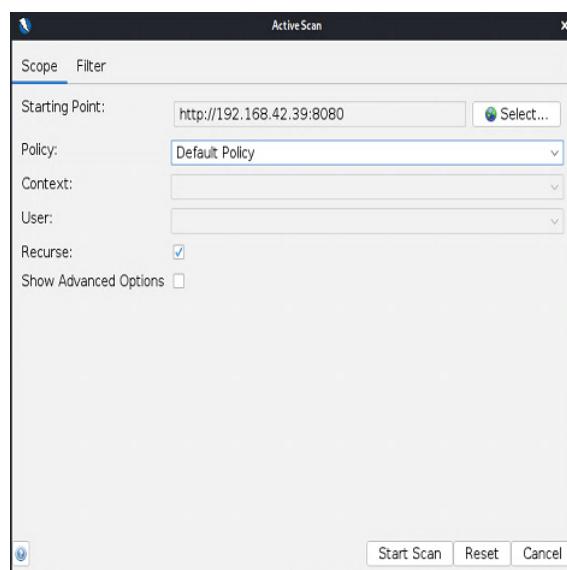


Figura 3.8: OWASP ZAP Active Scan Start

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”. en nuestro caso generaremos el tipo de Reporte **HTML** como se muestra en la figura 3.9:

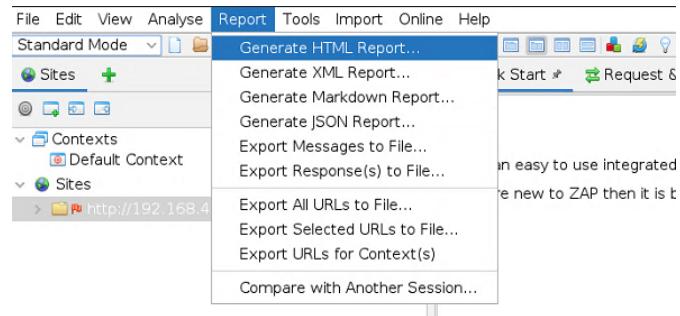


Figura 3.9: Reporte OWASP ZAP

Y seleccionamos uno de los tipos de reporte:

- HTML
- 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.

3.1.5. 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

3.2. 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 las 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 SonarScanner 4.6.2	SonarQube 8.2 PostgreSQL 13.3

Tabla 3.1: Infraestructura de pruebas

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

Para levantar el entorno ejecutamos:

```
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      | PostgreSQL Database directory appears to contain a database; Skipping initialization
db_1      | 2021-05-27 10:40:07.535 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 3.10: Docker compose up

3. Diseño solución técnica.

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

```
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.CePluginRepository] Load plugins
sonarqube_1 | 2021.05.27 10:44:51 INFO ce[]|o.s.c.CeComputeEngineContainerImpl] Running Community edition
sonarqube_1 | 2021.05.27 10:44:51 INFO ce[]|o.s.ce.app.CeServer] Compute 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 3.11: SonarQube server running

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

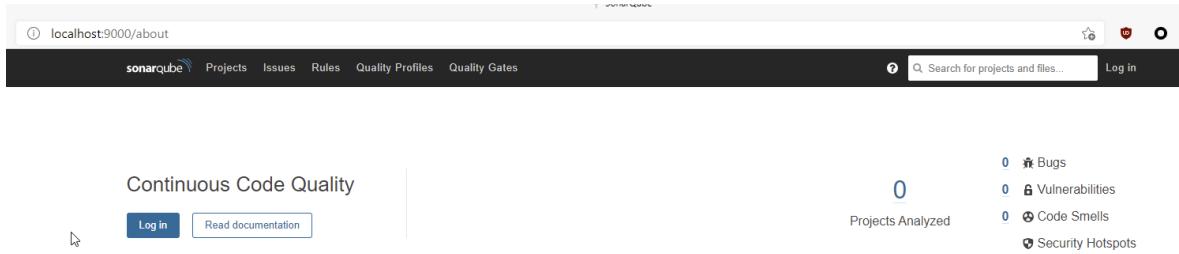


Figura 3.12: SonarQube portal

El fichero del docker compose [3](#) muestra los componentes de la arquitectura utilizados:

```
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:
  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 hemos elegido una aplicación para cada una de las tecnologías más representativas en el desarrollo web a largo de estos años:

- Java
- PHP
- JavaScript

Las aplicaciones que analizaremos serán las siguientes:

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

Tabla 4.1: Parámetros línea comandos dependency-check

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: Configuración plugin Dependency-Check

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: Propiedades plugin Dependency-Check

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:

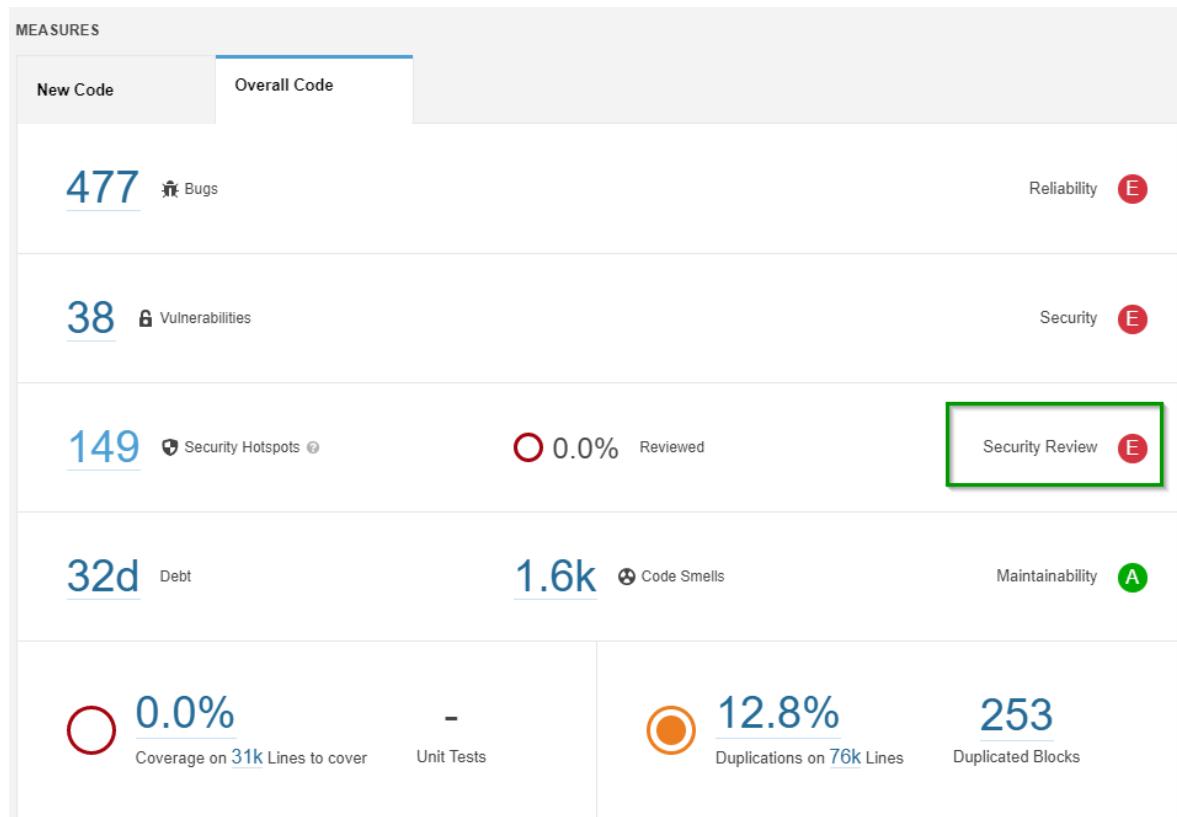


Figura 4.1: Resultado análisis estatico código WebGoat

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](#).

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](#) 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

Figura 4.2: WebGoat run docker compose

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

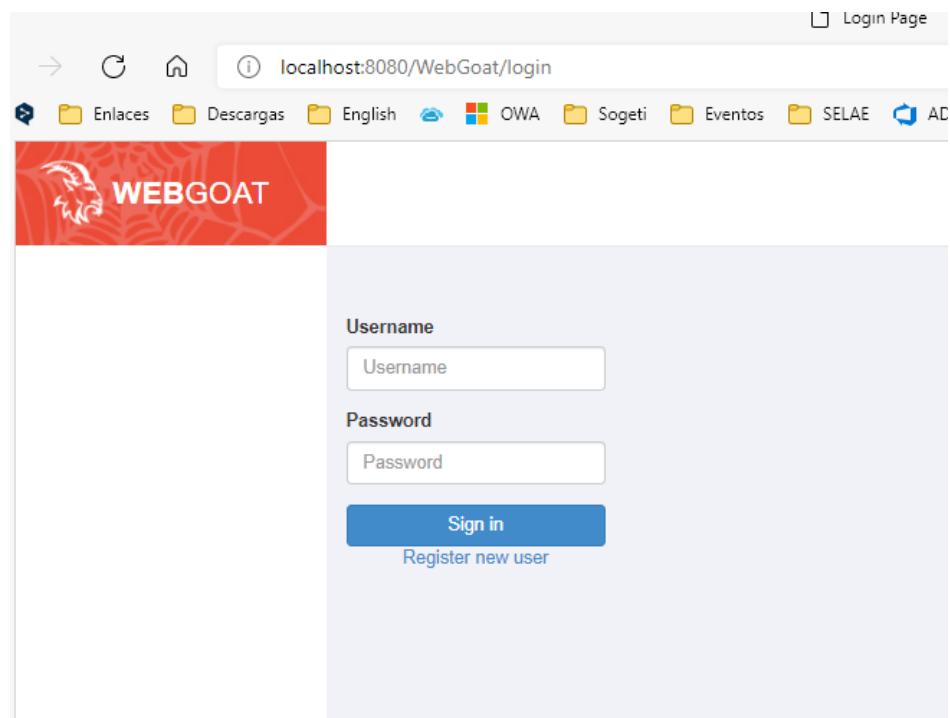
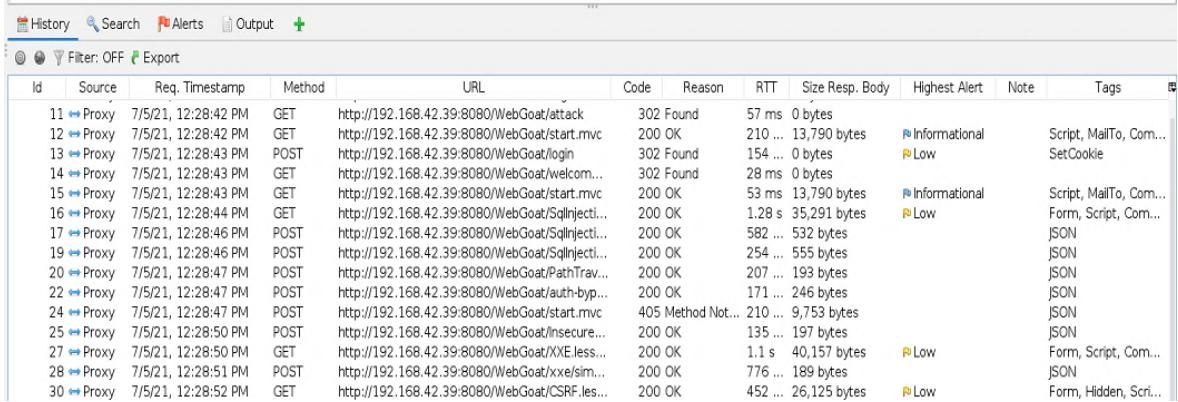


Figura 4.3: WebGoat run docker compose

4. Ejecución casos de prueba

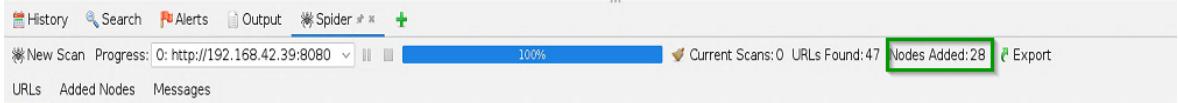
Una vez verificado, lanzamos el script de pruebas y revisamos que se vayan capturando las peticiones en OWASP ZAP correctamente



History												
Id	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size	Resp. Body	Highest Alert	Note	Tags
11	Proxy	7/5/21, 12:28:42 PM	GET	http://192.168.42.39:8080/WebGoat/attack	302	Found	57 ms	0 bytes		P Informational	Script, MailTo, Com...	
12	Proxy	7/5/21, 12:28:42 PM	GET	http://192.168.42.39:8080/WebGoat/start.mvc	200	OK	210 ...	13,790 bytes		P Low	SetCookie	
13	Proxy	7/5/21, 12:28:43 PM	POST	http://192.168.42.39:8080/WebGoat/login	302	Found	154 ...	0 bytes		P Low		
14	Proxy	7/5/21, 12:28:43 PM	GET	http://192.168.42.39:8080/WebGoat/welcome	302	Found	28 ms	0 bytes		P Low		
15	Proxy	7/5/21, 12:28:43 PM	GET	http://192.168.42.39:8080/WebGoat/start.mvc	200	OK	53 ms	13,790 bytes		P Informational	Script, MailTo, Com...	
16	Proxy	7/5/21, 12:28:44 PM	GET	http://192.168.42.39:8080/WebGoat/Sqliinjecti...	200	OK	1.28 s	35,291 bytes		P Low	Form, Script, Com...	
17	Proxy	7/5/21, 12:28:46 PM	POST	http://192.168.42.39:8080/WebGoat/Sqliinjecti...	200	OK	582 ...	532 bytes		P Low	JSON	
19	Proxy	7/5/21, 12:28:46 PM	POST	http://192.168.42.39:8080/WebGoat/Sqliinjecti...	200	OK	254 ...	555 bytes		P Low	JSON	
20	Proxy	7/5/21, 12:28:47 PM	POST	http://192.168.42.39:8080/WebGoat/PathTrav...	200	OK	207 ...	193 bytes		P Low	JSON	
22	Proxy	7/5/21, 12:28:47 PM	POST	http://192.168.42.39:8080/WebGoat/auth-byp...	200	OK	171 ...	246 bytes		P Low	JSON	
24	Proxy	7/5/21, 12:28:47 PM	POST	http://192.168.42.39:8080/WebGoat/start.mvc	405	Method Not... ...	210 ...	9,753 bytes		P Low	JSON	
25	Proxy	7/5/21, 12:28:50 PM	POST	http://192.168.42.39:8080/WebGoat/insecure...	200	OK	135 ...	197 bytes		P Low	JSON	
27	Proxy	7/5/21, 12:28:50 PM	GET	http://192.168.42.39:8080/WebGoat/XSSless...	200	OK	1.1 s	40,157 bytes		P Low	Form, Script, Com...	
28	Proxy	7/5/21, 12:28:51 PM	POST	http://192.168.42.39:8080/WebGoat/xss/sim...	200	OK	776 ...	189 bytes		P Low	JSON	
30	Proxy	7/5/21, 12:28:52 PM	GET	http://192.168.42.39:8080/WebGoat/CSRF.les...	200	OK	452 ...	26,125 bytes		P Low	Form, Hidden, Scri...	

Figura 4.4: WebGoat Zap Proxy Requests

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:



Spider												
New Scan	Progress:	0: http://192.168.42.39:8080	▼	100%	Current Scans: 0	URLs Found: 47	Nodes Added: 28	Export				
URLs	Added Nodes	Messages										

Figura 4.5: WebGoat Spider Result

Una vez finalizado el escáner en la ventana de progreso

http://192.168.42.39:8080 Scan Progress						
Host:	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:00:371	15		
Plugin						
Path Traversal	Medium		00:31.015	694	0	✓
Remote File Inclusion	Medium		00:16.957	420	0	✓
Source Code Disclosure - /WEB-INF folder	Medium		00:00.008	0	0	✗
External Redirect	Medium		00:14.642	378	0	✓
Server Side Include	Medium		00:10.932	168	0	✓
Cross Site Scripting (Reflected)	Medium		00:09.810	168	0	✓
Cross Site Scripting (Persistent) - Prime	Medium		00:12.214	42	0	✓
Cross Site Scripting (Persistent) - Spider	Medium		00:04.936	54	0	✓
Cross Site Scripting (Persistent)	Medium		00:05.338	0	0	✓
SQL Injection	Medium		00:26.912	940	1	✓
Server Side Code Injection	Medium		00:14.271	336	0	✓
Remote OS Command Injection	Medium		00:47.074	1344	0	✓
Directory Browsing	Medium		00:05.117	54	0	✓
Buffer Overflow	Medium		00:06.987	42	0	✓
Format String Error	Medium		00:15.522	126	2	✓
ORLF Injection	Medium		00:14.073	294	0	✓
Parameter Tampering	Medium		00:06.942	67	1	✓
ELMAH Information Leak	Medium		00:00.009	1	0	✓
.htaccess Information Leak	Medium		00:01.508	14	0	✓
Script Active Scan Rules	Medium		00:00.001	0	0	✗
Cross Site Scripting (DOM Based)	Medium		07:36.229	1130	0	✓
SOAP Action Spoofing	Medium		00:01.301	0	0	✓
SOAP XML Injection	Medium		00:05.499	0	0	✓
Totals			11:48.799	6367	4	

Figura 4.6: WebGoat Active Scan Finished

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

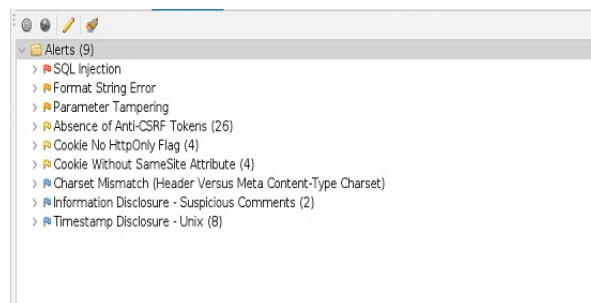


Figura 4.7: WebGoat Active Scan Alerts

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:

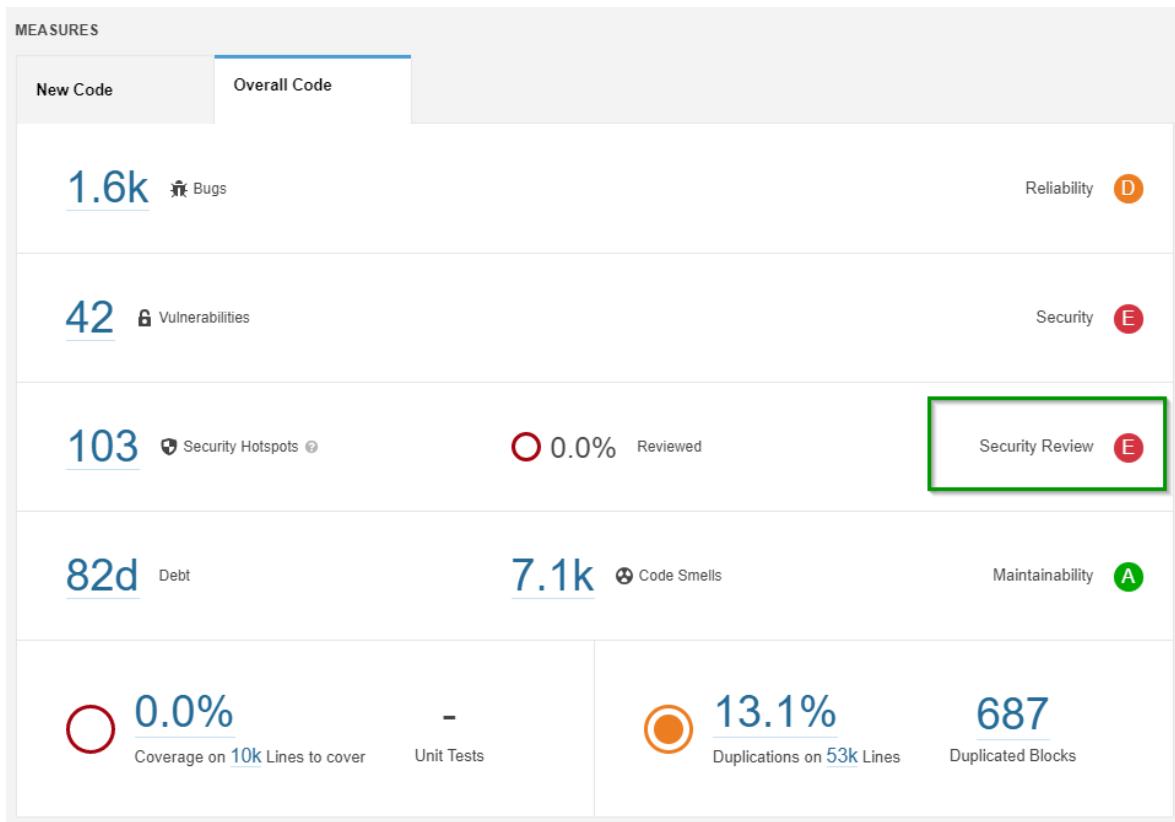


Figura 4.8: Resultado análisis estatico código DVWA

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:

The screenshot shows a security analysis tool interface. At the top, there are filters: 'Assigned to me' (selected), 'All', 'Status: To review', 'Overall code', and 'Security Hotspots Reviewed 0.0%'. On the left, a sidebar lists 103 security hotspots categorized by priority: HIGH (SQL Injection, Cross-Site Scripting (XSS)), MEDIUM (Denial of Service (DoS), Code Injection (RCE), Weak Cryptography), and LOW (Insecure Configuration, Log Injection, Object Injection). SQL Injection is currently selected. The main panel displays a detailed view of the SQL Injection rule (CVE-2017-5611). It explains that the rule flags the execution of SQL queries built using string formatting, even if no injection is present. It lists detected functions: mysql_query, mysql_db_query, mysql_unbuffered_query, pg_query, pg_send_query, mssql_query, mysqli_query and mysqli::query, mysqli_real_query and mysqli::real_query, mysqli_multi_query and mysqli::multi_query, mysqli_send_query and mysqli::send_query, PDO::query, PDO::exec, and PDO::prepare. Below this, an 'Exceptions' section shows code examples where concatenation does not raise issues. At the bottom, an 'Activity' section shows a recent update from June 3, 2021, at 12:50 AM, and a 'Add Comment' button.

Figura 4.9: DVWA Security hotspot

Todos los detalles de cada una de las vulnerabilidades, así como los “hotspot” están detalladas en el reporte de [análisis estático de código](#).

Con los defectos encontrados con el análisis estático más la información que el pentester vaya recolectando sobre la aplicación analizada se procede a crear un [plan de prueba](#) para la aplicación para poder capturar los hallazgos en la herramienta de análisis dinámico (OWASP ZAP).

4. Ejecución casos de prueba

Antes de ejecutar el plan de pruebas levantamos la aplicación levantando compilando el **contenedor** incluido en las fuentes 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>

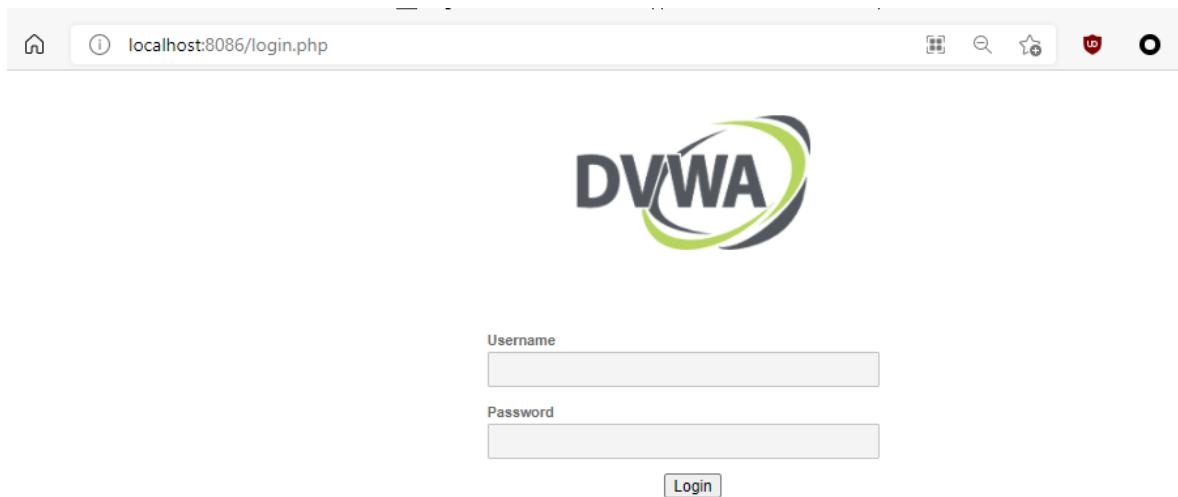


Figura 4.10: DVWA Application running

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

OWASP ZAP Requests												
Id	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp.	Body	Highest Alert	Note	Tags
366	Proxy	7/5/21, 8:04:38 PM	GET	http://192.168.42.39:8086/security.php	200	OK	5 ms	5,269 bytes	Medium	Form, Hidden, Script		
367	Proxy	7/5/21, 8:04:38 PM	GET	http://192.168.42.39:8086/	200	OK	6 ms	6,598 bytes	Medium	Script		
368	Proxy	7/5/21, 8:04:38 PM	POST	http://192.168.42.39:8086/vulnerabilities/exec/	200	OK	3.02 s	5,460 bytes	Medium	Form, Script		
369	Proxy	7/5/21, 8:04:41 PM	GET	http://192.168.42.39:8086/vulnerabilities/upload/	200	OK	12 ms	4,080 bytes	Medium	Form, Hidden, Uplo...		
370	Proxy	7/5/21, 8:04:48 PM	POST	http://192.168.42.39:8086/vulnerabilities/upload/	200	OK	10 ms	4,145 bytes	Medium	Form, Hidden, Uplo...		
371	Proxy	7/5/21, 8:04:53 PM	GET	http://192.168.42.39:8086/hackable/uploads/h...	404	Not Found	6 ms	277 bytes				
372	Proxy	7/5/21, 8:05:53 PM	GET	http://192.168.42.39:8086/hackable/uploads/h...	404	Not Found	6 ms	277 bytes				
373	Proxy	7/5/21, 8:06:25 PM	GET	http://192.168.42.39:8086/	302	Found	6 ms	0 bytes	Low	SetCookie		
374	Proxy	7/5/21, 8:06:25 PM	GET	http://192.168.42.39:8086/login.php	200	OK	9 ms	1,415 bytes	Medium	Form, Password, Hi...		
375	Proxy	7/5/21, 8:06:25 PM	POST	http://192.168.42.39:8086/login.php	302	Found	8 ms	0 bytes				
376	Proxy	7/5/21, 8:06:25 PM	GET	http://192.168.42.39:8086/index.php	200	OK	11 ms	6,685 bytes	Medium	Script		
377	Proxy	7/5/21, 8:06:25 PM	GET	http://192.168.42.39:8086/security.php	200	OK	7 ms	5,269 bytes	Medium	Form, Hidden, Script		
378	Proxy	7/5/21, 8:06:25 PM	GET	http://192.168.42.39:8086/	200	OK	9 ms	6,598 bytes	Medium	Script		
379	Proxy	7/5/21, 8:06:25 PM	POST	http://192.168.42.39:8086/vulnerabilities/exec/	200	OK	3.02 s	5,460 bytes	Medium	Form, Script		
380	Proxy	7/5/21, 8:06:28 PM	GET	http://192.168.42.39:8086/vulnerabilities/upload/	200	OK	8 ms	4,080 bytes	Medium	Form, Hidden, Uplo...		
381	Proxy	7/5/21, 8:06:30 PM	POST	http://192.168.42.39:8086/vulnerabilities/upload/	200	OK	8 ms	4,145 bytes	Medium	Form, Hidden, Uplo...		
382	Proxy	7/5/21, 8:06:33 PM	GET	http://192.168.42.39:8086/hackable/uploads/p...	200	OK	20 ms	89,671 bytes	Medium	Form, Hidden, Uplo...		
383	Proxy	7/5/21, 8:06:43 PM	GET	http://192.168.42.39:8086/vulnerabilities/sql/	200	OK	11 ms	4,207 bytes	Medium	Form, Script		

Figura 4.11: DVWA Zap Proxy Requests

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:

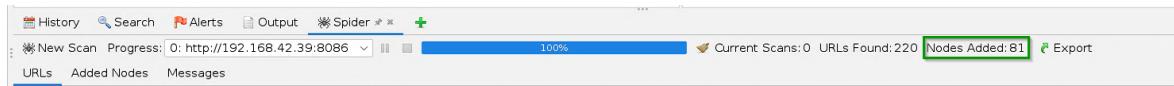


Figura 4.12: WebGoat Spider Result

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:

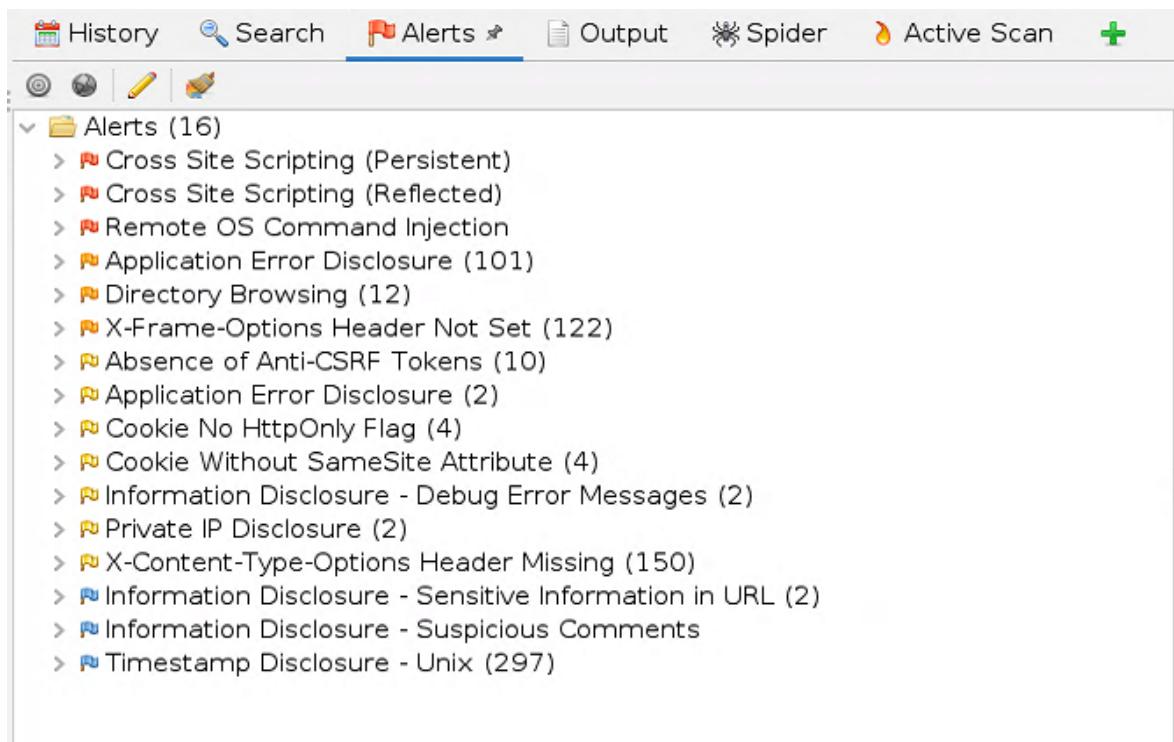


Figura 4.13: DVWA Active Scan Alerts

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:

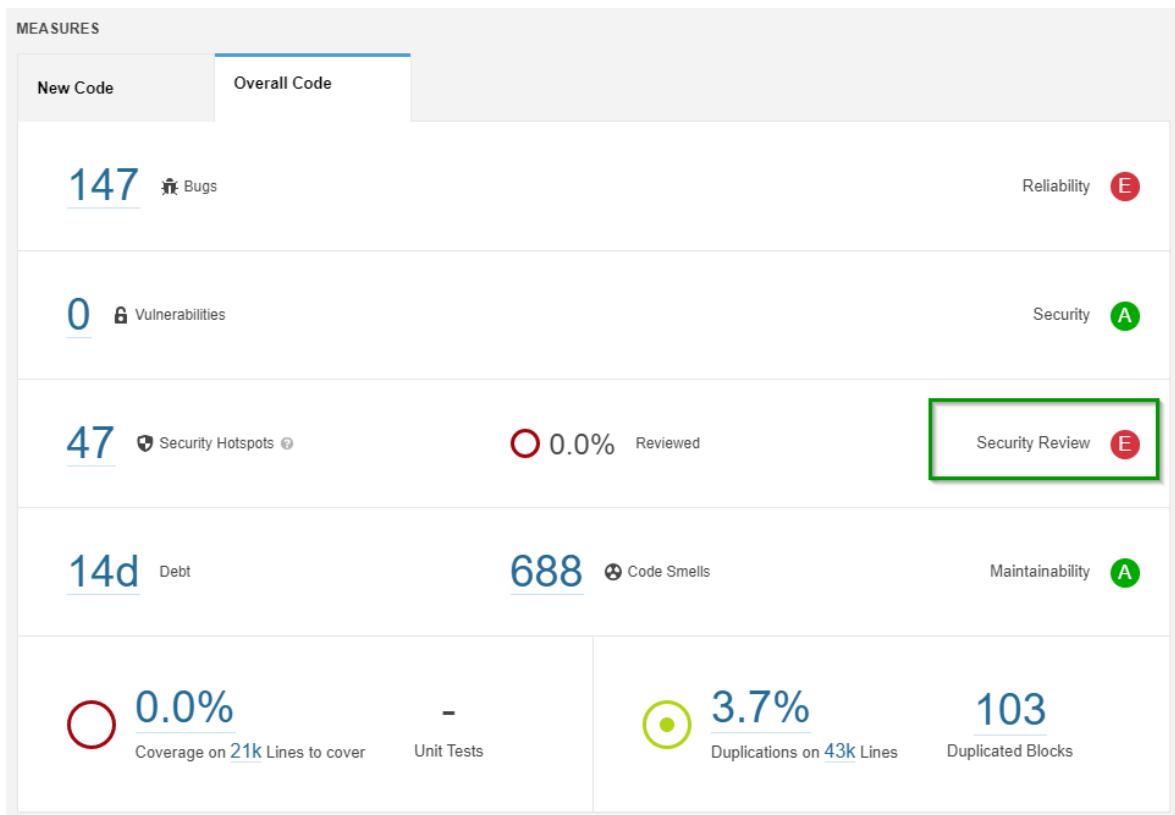


Figura 4.14: Resultado análisis estatico código Juice Shop

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](#).

Con los defectos encontrados con el análisis estático más la información que el pentester vaya recolectando sobre la aplicación analizada se procede a crear un [plan de prueba](#) 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>

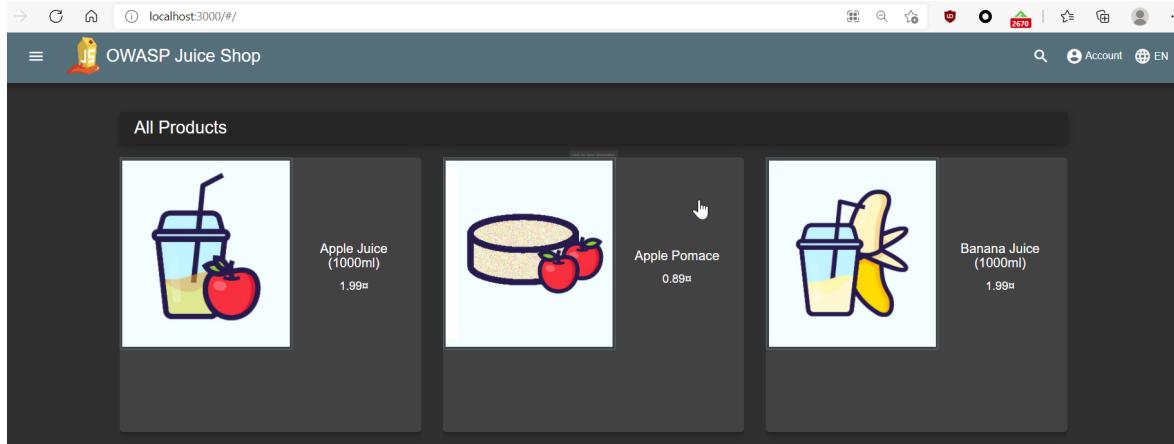


Figura 4.15: JuiceShop Application running

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

History												
Filter: OFF												
Output												
Id	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size	Resp. Body	Highest Alert	Note	Tags
20 ↳ Proxy	7/6/21, 8:15:15 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	34 ms	0 bytes				Medium		
21 ↳ Proxy	7/6/21, 8:15:15 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	25 ms	0 bytes				Medium		
23 ↳ Proxy	7/6/21, 8:15:15 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	95 ms	0 bytes				Medium		
36 ↳ Proxy	7/6/21, 8:15:16 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	13 ms	0 bytes				Medium		
46 ↳ Proxy	7/6/21, 8:16:31 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	12 ms	0 bytes				Medium		
47 ↳ Proxy	7/6/21, 8:16:42 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	26 ms	0 bytes				Medium		
61 ↳ Proxy	7/6/21, 8:18:39 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	22 ms	0 bytes				Medium		
101 ↳ Proxy	7/6/21, 8:23:51 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	17 ms	0 bytes				Medium		
152 ↳ Proxy	7/6/21, 8:29:58 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	28 ms	0 bytes				Medium		
155 ↳ Proxy	7/6/21, 8:29:58 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	61 ms	0 bytes				Medium		
158 ↳ Proxy	7/6/21, 8:29:58 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	246 ms	0 bytes				Medium		
166 ↳ Proxy	7/6/21, 8:29:59 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	326 ms	0 bytes				Medium		
174 ↳ Proxy	7/6/21, 8:30:05 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	22 ms	0 bytes				Medium		
201 ↳ Proxy	7/6/21, 8:48:01 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	28 ms	0 bytes				Medium		
317 ↳ Proxy	7/6/21, 9:35:00 AM	GET	http://192.168.2.130:3000/rest/admin/application...	200 OK	215 ms	17,625 bytes				Medium		
328 ↳ Proxy	7/6/21, 9:35:32 AM	GET	http://192.168.2.130:3000/rest/admin/application...	200 OK	20 ms	17,625 bytes				Medium	JSON	
332 ↳ Proxy	7/6/21, 9:35:39 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	86 ms	0 bytes				Medium	JSON	
338 ↳ Proxy	7/6/21, 9:35:40 AM	GET	http://192.168.2.130:3000/rest/admin/application...	200 OK	71 ms	17,625 bytes				Medium	JSON	
353 ↳ Proxy	7/6/21, 9:36:28 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	67 ms	0 bytes				Medium		
357 ↳ Proxy	7/6/21, 9:36:28 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	179 ms	0 bytes				Medium		
364 ↳ Proxy	7/6/21, 9:36:29 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	178 ms	0 bytes				Medium		
370 ↳ Proxy	7/6/21, 9:36:29 AM	GET	http://192.168.2.130:3000/rest/admin/application...	304 Not Modified	45 ms	0 bytes				Medium		
425 ↳ Proxy	7/6/21, 10:01:10 AM	GET	http://192.168.2.130:3000/rest/admin/application...	200 OK	34 ms	17,625 bytes				Medium	JSON	
14 ↳ Proxy	7/6/21, 0:15:15 AM	GET	http://192.168.2.130:3000/rest/admin/application...	200 OK	92 ms	20 bytes				Medium	JSON	

Figura 4.16: JuiceShop Zap Proxy Requests

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:

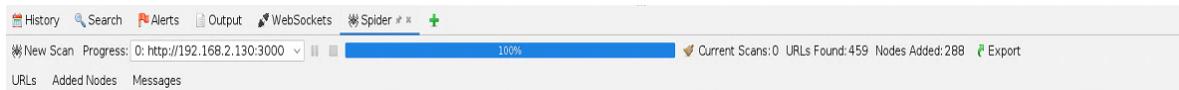


Figura 4.17: JuiceShop Spider Result

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:

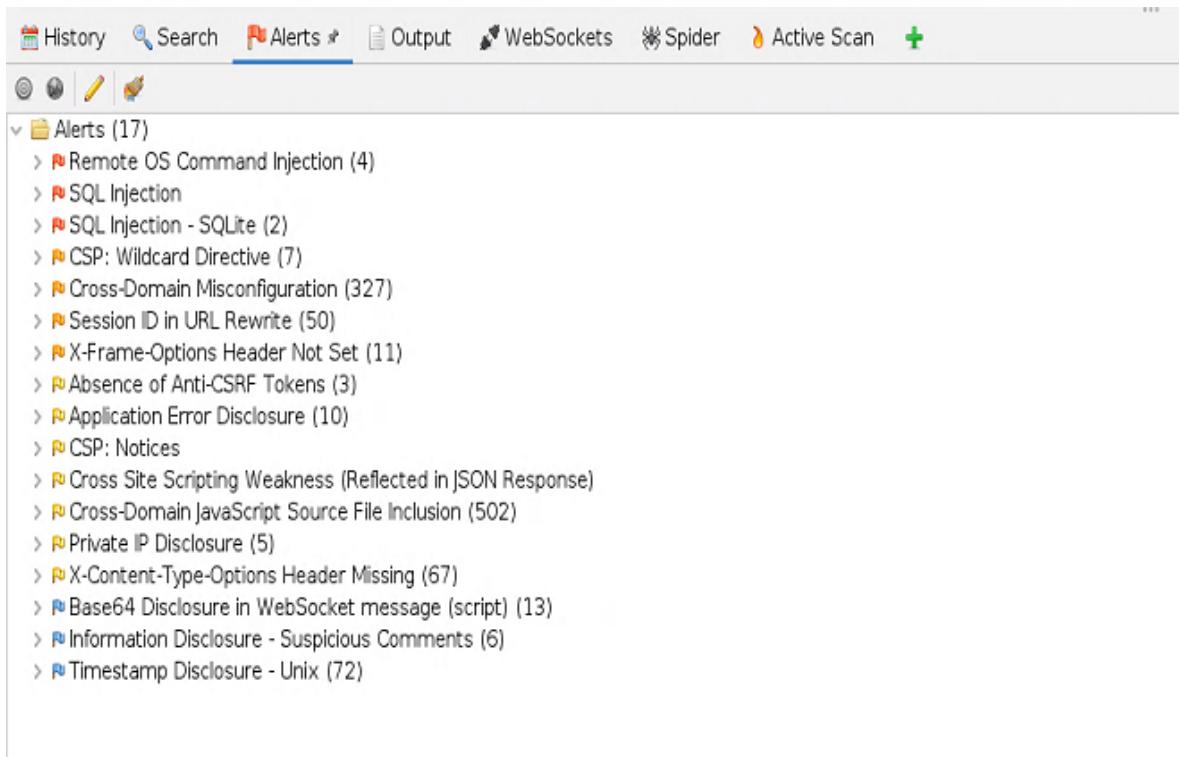


Figura 4.18: JuiceShop Active Scan Alerts

5. Conclusões y trabajo futuro

5.1. 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 defecto 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.

5.2. Trabajo futuro

Uno paso lógico, para continuar en la línea del proyecto, sería integrar en proceso descrito en el presente PFG en una [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
- Jenkin
- 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.

A. Anexo 1 - Informes generados

Como se vio en la sección 3.1.5 se han generado los reportes requeridos para cada aplicación.

A.0.1. Definición del plan de pruebas de seguridad

- PPR WebGoat - Plan Pruebas de Seguridad.docx
- PPR DVWA - Plan Pruebas de Seguridad.docx
- PPR JuiceShop - Plan Pruebas de Seguridad.docx

A.0.2. Reporte análisis estático de código

- ReporteAnalisisestatico_{WebGoat}.docx
- ReporteAnalisisestatico_{DVWA}.docx
- ReporteAnalisisestatico_{JuiceShop}.docx

A.0.3. Plan pruebas para el análisis dinámico

- PlanPruebas_OWASPWebGoat.ps1
- PlanPruebas_DVWA.ps1
- PlanPruebas_OWASPJuiceShop.ps1

A.0.4. Reporte análisis dinámico

- WebGoat- ReporteAnalisisDinamico.html
- DVWA - ReporteAnalisisDinamico.html
- JuiceShop - ReporteAnalisisDinamico.html

A.0.5. Informe resultado ejecución pruebas de seguridad

- [WebGoat - Informe Resultados Pruebas de Seguridad.docx](#)
- [DVWA - Informe Resultados Pruebas de Seguridad.docx](#)
- [JuiceShop - Informe Resultados Pruebas de Seguridad.docx](#)

Referencia bibliográfica

- [Gonzalez, 2013] Gonzalez, P. (2013). *Pentesting con Kali*. OxWord.
- [Navarro, 2021] Navarro, E. J. R. (2021). Repositorio pfg. <https://github.com/M0l1n3ta/PFG/tree/master/Reportes/PPR%20-%20Plan%20de%20pruebas>.
- [OWASP, 2017] OWASP (2017). Owasp testing project. <https://owasp.org/www-project-web-security-testing-guide/stable/2-Introduction/README.html#Testing-Techniques-Explained>.
- [OWASP, 2019] OWASP (2019). Owasp asvs 4.0. <https://owasp.org/www-project-application-security-verification-standard/>.

Glosario

A

ASVS

El proyecto OWASP Application Security Verification Standard (ASVS) proporciona una base para realizar los controles de seguridad técnicos en aplicaciones web, además también proporciona un listado de requisitos a cumplir para un desarrollo seguro.. [18](#), [55](#)

C

CD

Del inglés "Continuous Deployment", término que hace referencia a la implantación de procesos automáticos de despliegue de las aplicaciones en el ciclo de vida del desarrollo de software.. [18](#), [55](#)

CI

Del inglés "Continuous Integration", término que hace referencia a la implantación de procesos automáticos de compilación y revisión del código fuente en el ciclo de vida del desarrollo software.. [18](#), [55](#)

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](#)" con fondos de la "National Cyber Security Division" ([NVD](#)) del gobierno de los Estados Unidos de América.. [22](#), [55](#)

D

DAST

Del inglés "Dynamic Application Security Testing", término que hace referencia a las pruebas de análisis dinámicas de código.. [18](#), [55](#)

defacement

Del inglés "*desfiguración*", es un ataque a un sitio web que cambia la apariencia visual de una página web.. [14](#), [55](#)

E

exploit

Término inglés 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.. [6](#), [55](#)

O

OWASP

El Open Web Application Security Project (OWASP) es una comunidad mundial libre y abierta enfocada en mejorar la seguridad del desarrollo de software. [18](#), [55](#)

P

parser

Un parser es un compilador o intérprete que divide los datos en elementos más pequeños para traducirlos fácilmente a otro lenguaje. Un parser toma la entrada en forma de una secuencia de tokens, comandos interactivos o instrucciones de programa y los divide en partes que pueden ser utilizadas por otros componentes en la programación.. [12](#), [55](#)

pipeline

Una pipeline es una nueva forma de trabajar en el mundo devops en la integración y despliegue continuos. Mediante una pipeline, podemos definir el ciclo de vida completo de una aplicación (descargar código, compilar, test, desplegar, etc.) mediante código.. [52](#), [55](#)

S

SAST

Del inglés "Static Application Security Testing", término que hace referencia a las pruebas de análisis estático de código. [18](#), [55](#)

SCA

Del inglés "Static Code Analysis", término que hace referencia a las pruebas de análisis estático de código. [18](#), [55](#)

SSDLC

Del inglés "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 del software, tales como pruebas de penetración, revisiones de código o análisis de dependencias.. [18](#), [55](#)

SUT

Del inglés "System Under Test", término que hace referencia a la aplicación o sistema sobre el cual se ejecutarán las pruebas.. [6](#), [55](#)