Universitat Oberta de Catalunya

uoc.edu

“Integración de analizadores automáticos de código”

Alumno: Urko Fernández Román
Plan de Estudios: Máster Universitario en Ciberseguridad y Privacidad (MUCYP)
Área del trabajo final: Análisis de datos

Nombre del consultor: Joan Caparrós Ramírez
Profesora responsable de la asignatura: Cristina Pérez Solà

Fecha Entrega: 1 de junio de 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Integración de Analizadores automáticos de código*
Nombre del autor:	*Urko Fernández Román*
Nombre del consultor/a:	*Joan Caparrós Ramírez*
Nombre del PRA:	*Cristina Pérez Solà*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Análisis de datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*OWASP, SDLC, bugs*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El presente trabajo tiene como finalidad estudiar distintas herramientas SAST,DAST,IAST y SCA mencionadas en la web de OWASP. Se seleccionan tres diferentes complementarias entre sí y proponiendo su integración dentro del ciclo de desarrollo y despliegue de una web basada en WordPress y construida con componentes de código abierto por un pequeño equipo. El objetivo es encontrar una solución que no castigue el ritmo de trabajo durante el proceso de desarrollo ni requiera mayor esfuerzo de aprendizaje, y consiga reducir significativamente el número de vulnerabilidades.

Para alcanzar el objetivo, el trabajo se divide en dos partes, analizando primero las partes fuertes y débiles, funcionamiento y la forma en que presentan sus resultados varias herramientas. Seguidamente, se procede a integrar las herramientas seleccionadas en el SDLC, implementando una metodología para reducir falsos positivos a través de una estrategia pragmática de aprendizaje de buenas prácticas de seguridad de forma iterativa.

La solución propuesta está compuesta íntegramente por herramientas de código abierto amadrinadas por la propia OWASP: ASST (SAST), ZAP (DAST) y Dependency-Check (SCA). Se propone también cómo integrar cada herramienta en las distintas fases y cómo aprovechar sus resultados en un proceso de mejora continua. A modo de ejemplo, se muestran los resultados de análisis sobre una web real desarrollada hace año y medio en WordPress.

A pesar de las evidentes diferencias en cuanto funcionalidad e integración entre herramientas comerciales potentes y soluciones libres, éstas se han mostrado capaces de adaptarse al contexto con gran potencial de mejora y ofreciendo oportunidades de aprendizaje.

Abstract (in English, 250 words or less):

This essay aims to study different SAST, DAST, IAST and SCA tools mentioned in the OWASP website. Three different complementing tools are selected, offering a way to integrate them within the development and deployment cycle of a WordPress-based website built with open source components by a small team. The goal is to find a solution that doesn't impact the workflow of the development process, doesn't require a high learning curve, and significantly reduces the number of vulnerabilities.

To achieve the objective, the work is divided into two parts, first analysing the strengths and weaknesses, performance and the way in which various tools present their results. We then proceed to integrate the selected tools into the SDLC, implementing a methodology to reduce false positives through a pragmatic strategy of learning good security practices in an iterative way.

The proposed solution is composed entirely of open source tools backed by OWASP itself: ASST (SAST), ZAP (DAST) and Dependency-Check (SCA). There's also a proposal of how to integrate each tool in the different phases and how to take advantage of their results in a process of continuous improvement. As an example, the results of analyses on a real website developed a year and a half ago in WordPress are shown.

Despite the obvious differences in functionality and integration capabilities between powerful commercial tools and these open source solutions, the latter have proven to be able to adapt to the context with great potential for improvement and learning opportunities.

Índice

Índice de ilustraciones

.1. Introducción

.1.1 Contexto y justificación

En un contexto en el que la mayoría de los incidentes de seguridad son fruto de la explotación de fallos de software (se estima que el 90% de los incidentes de seguridad notificados se deben a la explotación de defectos en el diseño o en el código del software[1]), es de vital importancia evitar y corregir errores de programación y diseño durante el proceso de desarrollo a fin de garantizar un contexto de mayor seguridad para cuando las aplicaciones estén en funcionamiento.

Mientras que la seguridad de la red ha mejorado y se ha ido reforzando año tras año[2], no ocurre igual con la seguridad de las aplicaciones, a pesar de que la mayoría de los ciber-ataques ocurren a nivel de aplicación[1]. El problema reside en que ingenieros e ingenieras de seguridad de la información no entienden el desarrollo de software y la mayoría de la gente que se dedica al desarrollo de software no entiende la ciber-seguridad[3].

Debido a las exigencias de mercado, muchas organizaciones y equipos de desarrollo se apresuran por lanzar cuanto antes una aplicación web o móvil, en lugar de asegurarse de que el software es seguro, o dejan la securización para más adelante. Además de la programación no segura, esta situación fuerza a que muchas veces al programar se acabe copiando -sin darse cuenta- fallos de seguridad en el software, ya que es habitual pedir prestado el código a otras personas de forma rutinaria o copiarlo de algún foro, libro o tutorial, y normalmente no se comprueba este código en busca de fallos de seguridad.

Como consecuencia de las prisas, estas entidades acaban exponiendo datos sensibles propios y de la clientela a posibles amenazas externas, o se arriesgan a tener que asumir un mayor coste a la hora de corregir los fallos en la fase de mantenimiento[4]. El coste de corregir un error detectado después de la publicación del producto puede ser hasta cuatro a cinco veces mayor[5] que uno descubierto durante el diseño y desarrollo del mismo.

Para evitarlo, existen numerosas estrategias, metodologías y herramientas para ayudar a los equipos de desarrollo a encontrar y corregir los fallos de software antes de que los productos y servicios lleguen a estar disponibles para quienes estuvieran dirigidos. Varias herramientas que podemos incorporar a los procesos de desarrollo están basadas en distintas metodologías como son SAST[2],DAST[3],IAST[4] o SCA[5], que sirven para analizar el código fuente, los componentes integrados en el mismo, o el comportamiento de la aplicación en ejecución a fin de encontrar fallos y vulnerabilidades.

1 Séptimo nivel del modelo OSI y el cuarto de la pila TCP/IP
2 Static Application Security Testing, ver apartado 2.2
3 Dynamic Application Security Testing, ver apartado 2.3
4 Interactive Application Security Testing, ver apartado 2.4
5 Software Composition Analysys, ver apartado 2.5

La industria ha puesto recursos y el foco en el testeo de vulnerabilidades conocidas en aplicaciones que están ya publicadas y en funcionamiento, y se ha dedicado poca atención en reforzar el proceso de desarrollo para ir corrigiendo los fallos de programación y seguridad durante todas las fases del ciclo de desarrollo. La falta de recursos (económicos, temporales, de personal…) con las que muchos equipos de desarrollo afrontan sus proyectos, y el hecho de que normalmente quien se dedica al desarrollo de software no ha recibido formación sobre prácticas de programación seguras, obligan a la búsqueda de soluciones que se integren de forma sencilla y automaticen lo máximo posible el uso de este tipo de herramientas.

Este trabajo estudiará las partes fuertes y débiles de varias herramientas SAST, DAST, IAST o SCA mencionadas por la Open Source Foundation for Application Security (OWASP), una comunidad online que produce y pone a disposición libre artículos, metodologías, herramientas, documentación y diversas tecnologías relacionadas con la seguridad de las aplicaciones web. Una vez estudiado el funcionamiento de las herramientas (qué analizan, cómo, qué resultados ofrecen...), se propondrá la integración de algunas de ellas dentro del ciclo de desarrollo y despliegue de una aplicación web. La metodología propuesta se basará en herramientas libres o gratuitas para proyectos de código abierto, a fin de encontrar una combinación que ayuden a elevar el listón de calidad y seguridad del software con un coste de adquisición e implementación bajo.

Si bien es cierto que en el movimiento del software libre prima más la libertad y derechos de las personas que utilizan el software licenciado, frente al enfoque del campo del código abierto que se basa más en los beneficios prácticos[6], a lo largo de este documento se utilizará tanto el término “código abierto” como “software libre” indistintamente para referirse al mismo tipo de soluciones basadas en software que se distribuyen con licencias abiertas y permisivas con respecto al código fuente y el uso que se puede hacer del mismo.

.1.2 Objetivos del Trabajo

El objetivo general de este trabajo de fin de máster es analizar las distintas herramientas SAST, DAST, IAST y SCA mencionadas en la web de OWASP y proponer la integración de un conjunto de ellas dentro del ciclo de desarrollo y despliegue de una aplicación web.

El auge de las aplicaciones en la nube y la omnipresencia de los sistemas de gestión de contenidos (CMS[6]) de código abierto tipo Wordpress ha creado un contexto donde proliferan los sitios web vulnerables[7], situación especialmente preocupante cuando dos de cada cinco sitios web en Internet están construidos sobre este CMS[8].

Como no es posible considerar los contextos de tantos tipos de aplicaciones web, sobre todo aquellas de mayor dimensión o complejidad, este trabajo se enfocará en las mejores soluciones que se pudieran integrar a desarrollos web

6 Content Management System

pequeños y medianos que utilizan componentes de código abierto, siendo estos emprendimientos normalmente realizados por equipos compuestos por pocas personas.

Como objetivos específicos tendríamos los siguientes:

- Encontrar el equilibrio entre las herramientas que se deberían integrar en los procesos para detectar los fallos de seguridad, y el impacto que estos cambios tendrían en el flujo del ciclo de desarrollo.

- Obtener una selección de analizadores automáticos de código y de vulnerabilidades que puedan incorporarse al proceso de desarrollo de equipos pequeños que no cuenten con los recursos (ni temporales ni económicos) para realizar una auditoría fuerte de seguridad de código.

- Encontrar una combinación compatible y complementaria de las distintas estrategias que implementan las herramientas SAST, DAST, IAST o SCA para que la integración del desarrollo de la seguridad sea sin interrupciones y claramente beneficiosa para el equipo de desarrollo.

- Plantear una metodología que sirva para que un equipo pequeño de desarrollo pueda aprender buenas prácticas de programación segura con ayuda de las herramientas de asistencia que se seleccionarán.

.1.3 Enfoque y método seguido

Este proyecto se dividirá en dos partes principalmente. Por un lado, se realizará una investigación alrededor de las herramientas SAST, DAST, IAST y SCA más comunes que se recomiendan desde la web de OWASP. Para cada herramienta, se analizarán sus partes fuertes y débiles, qué parte del producto de software analizan, qué métodos utilizan para ello, y cuales son los resultados que se pueden obtener al usarlas.

Por otro lado, se implementará una propuesta de integración de las herramientas seleccionadas dentro del ciclo de desarrollo y despliegue de una aplicación web basada en componentes de código abierto. Más adelante veremos cómo haberse enfocado en este tipo de desarrollos de código abierto nos permitirá incorporar de forma asequible herramientas basadas en SCA para garantizar que en su conjunto el desarrollo web cuenta con menos vulnerabilidades.

La correcta interpretación del análisis que se realizará durante la primera parte será fundamental para poder proponer en la segunda parte el diseño del protocolo en el que se indicará cómo se incorporarán las distintas herramientas según la función necesaria, así como el coste y requerimientos que tendría hacerlo. Para cumplir con los objetivos marcados, es necesario que esta integración de analizadores de código no interrumpa ni requiera esfuerzo excepcional al equipo de desarrollo. Para ello, las herramientas seleccionadas

deben ser capaces de proporcionar información sencilla de interpretar a fin de aplicar las correcciones necesarias.

En la planificación del trabajo, se han tenido en cuenta días festivos, pero se estima que se podrán realizar algunas de las tareas en ellos, así como en fines de semana, en caso de que haya retrasos en algunas de ellas.

.1.4 Planificación del Trabajo

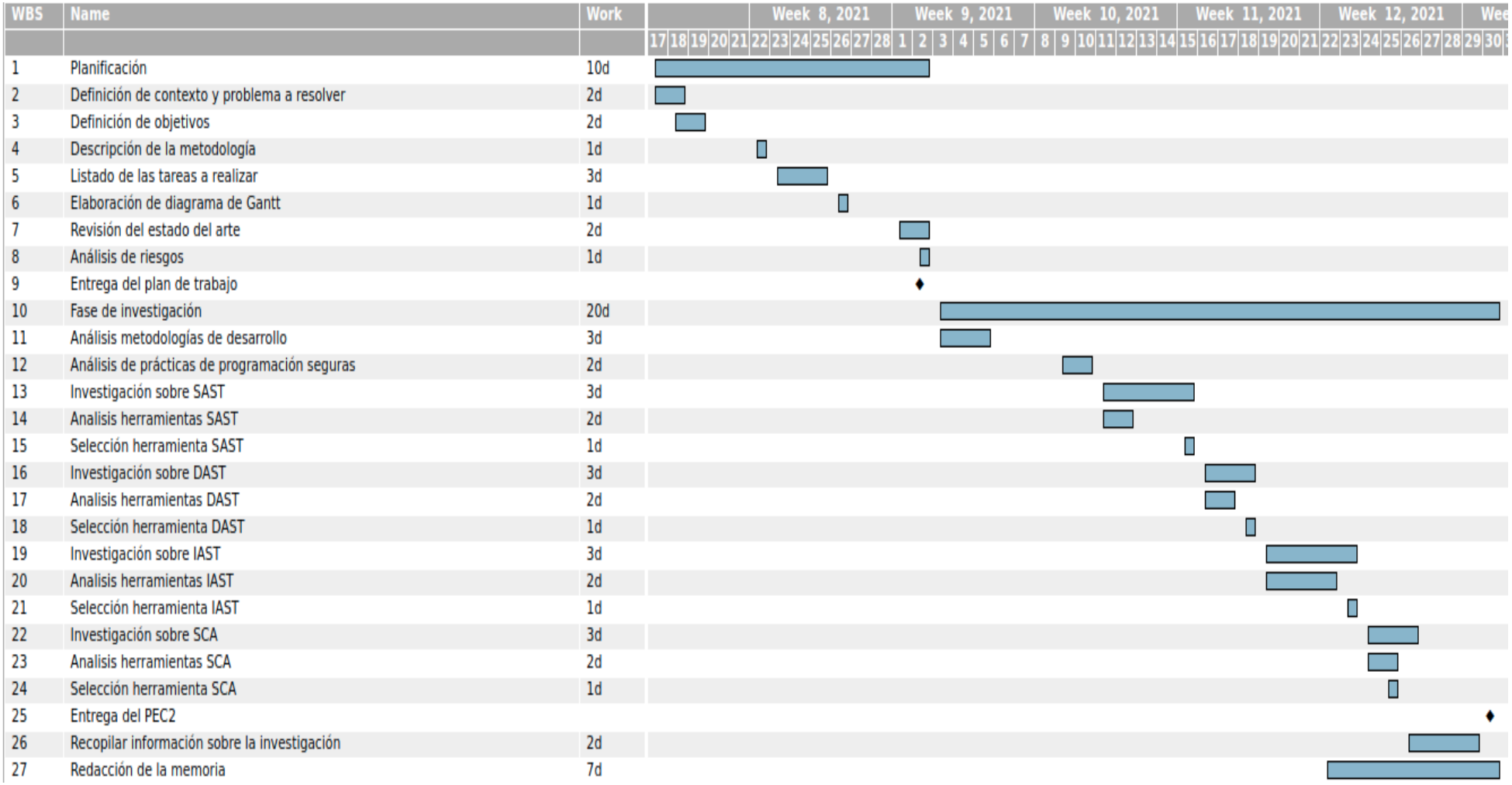
En el siguiente cuadro se muestran las distintas actividades de las que se compone este trabajo junto con la planificación temporal y duración estimada para ejecutarlas:

Código	Actividad	Inicio	Fin	Duración
1	Planificación			
1.1	Definición de contexto y problema a resolver	17/2/21	18/2/21	2
1.2	Definición de objetivos	18/2/21	19/2/21	2
1.3	Descripción de la metodología	22/2/21	22/2/21	1
1.4	Listado de las tareas a realizar	23/2/21	25/2/21	3
1.5	Elaboración de diagrama de Gantt	26/2/21	26/2/21	1
1.6	Revisión del estado del arte	1/3/21	2/3/21	2
1.7	Análisis de metodologías de desarrollo de software	3/3/21	5/3/21	3
1.8	Análisis de riesgos	2/3/21	2/3/21	1
1.9	**Entrega del plan de trabajo**		**2/3/21**	**Hito**
2	Fase de investigación			
2.1	Prácticas de programación seguras (OWASP)	9/3/21	10/3/21	2
2.2	Investigación sobre SAST	11/3/21	15/3/21	3
2.2.1	Analisis herramientas SAST	11/3/21	12/3/21	2
2.2.2	Selección herramientas SAST	15/3/21	15/3/21	1
2.3	Investigación sobre DAST	16/3/21	18/3/21	3
2.3.1	Analisis herramientas DAST	16/3/21	17/3/21	2
2.3.2	Selección herramientas DAST	18/3/21	18/3/21	1
2.4	Investigación sobre IAST	19/3/21	23/3/21	3
2.4.1	Analisis herramientas IAST	19/3/21	22/3/21	2
2.4.2	Selección herramientas IAST	23/3/21	23/3/21	1
2.5	Investigación sobre SCA	24/3/21	26/3/21	3
2.5.1	Analisis herramientas SCA	24/3/21	25/3/21	2
2.5.2	Selección herramientas SCA	25/3/21	25/3/21	1
2.6	**Entrega del PEC2**		**30/3/21**	**Hito**
2.6.1	Recopilar información sobre la investigación	26/3/21	29/3/21	2
2.6.2	Redacción de la memoria	22/3/21	30/3/21	5
3	Fase de implementación			
3.1	Integración herramientas en SDLC	31/3/21	22/4/21	

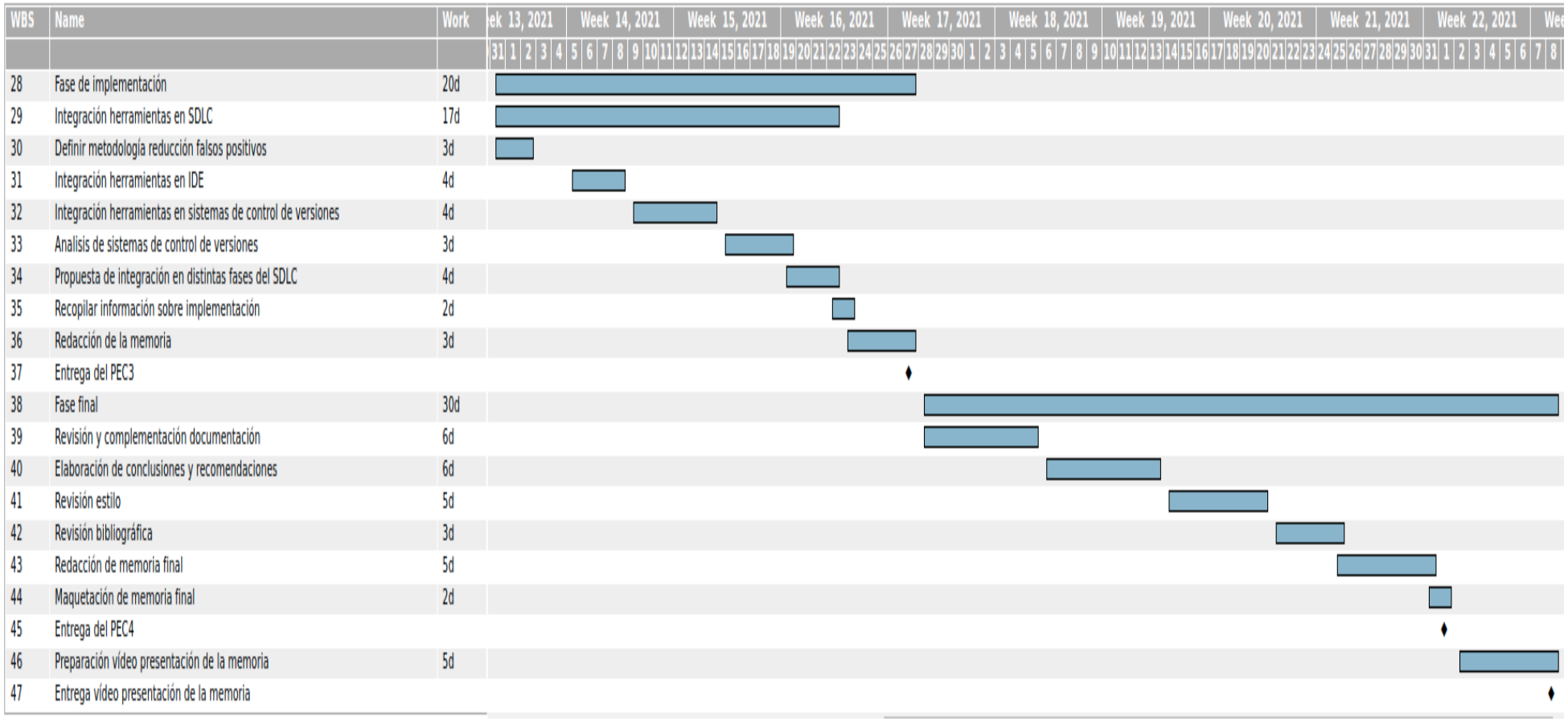
3.1.1	Metodología para reducir falsos positivos	31/3/21	2/4/21	3
3.1.2	Integración herramientas en IDE	5/4/21	8/4/21	4
3.1.3	Análisis de sistemas de control de versiones (Git, Subversion, Mercurial...)	9/4/21	13/4/21	3
3.1.4	Integración herramientas en sistemas de control de versiones	14/4/21	19/4/21	4
3.1.5	Propuesta de integración en ciclo de desarrollo y despliegue de una aplicación	19/4/21	22/4/21	4
3.2	**Entrega del PEC3**		**27/4/21**	**Hito**
3.2.1	Recopilar información sobre implementación	22/4/21	23/4/21	2
3.2.2	Redacción de la memoria	23/4/21	27/4/21	3
4	Fase final			
4.1	**Entrega del PEC4**		**1/6/21**	**Hito**
4.1.1	Revisión y complementación de la documentación	28/4/21	5/5/21	6
4.1.2	Elaboración de conclusiones y recomendaciones	6/5/21	13/5/21	6
4.1.3	Revisión estilo	14/5/21	20/5/21	5
4.1.4	Revisión bibliográfica	21/5/21	25/5/21	3
4.1.5	Redacción de memoria final	25/5/21	31/5/21	5
4.1.6	Maquetación de memoria final	31/5/21	1/6/21	**2**
4.2	**Preparación vídeo presentación de la memoria**	**2/6/21**	**8/6/21**	**Hito**

.1.5 Diagrama de Gantt

Se incluye a continuación la planificación temporal del trabajo mediante un diagrama de Gantt dividido en dos páginas. En la primera encontramos las fases de planificación e investigación, y en la segunda las fases de implementación y la fase final:



WBS	Name	Work
1	Planificación	10d
2	Definición de contexto y problema a resolver	2d
3	Definición de objetivos	2d
4	Descripción de la metodología	1d
5	Listado de las tareas a realizar	3d
6	Elaboración de diagrama de Gantt	1d
7	Revisión del estado del arte	2d
8	Análisis de riesgos	1d
9	Entrega del plan de trabajo	
10	Fase de investigación	20d
11	Análisis metodologías de desarrollo	3d
12	Análisis de prácticas de programación seguras	2d
13	Investigación sobre SAST	3d
14	Analisis herramientas SAST	2d
15	Selección herramienta SAST	1d
16	Investigación sobre DAST	3d
17	Analisis herramientas DAST	2d
18	Selección herramienta DAST	1d
19	Investigación sobre IAST	3d
20	Analisis herramientas IAST	2d
21	Selección herramienta IAST	1d
22	Investigación sobre SCA	3d
23	Analisis herramientas SCA	2d
24	Selección herramienta SCA	1d
25	Entrega del PEC2	
26	Recopilar información sobre la investigación	2d
27	Redacción de la memoria	7d


WBS
Name
Work
ek 13, 2021
Week 14, 2021
Week 15, 2021
Week 16, 2021
Week 17, 2021
Week 18, 2021
Week 19, 2021
Week 20, 2021
Week 21, 2021
Week 22, 2021
31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8
28 Fase de implementación 20d
29 Integración herramientas en SDLC 17d
30 Definir metodología reducción falsos positivos 3d
31 Integración herramientas en IDE 4d
32 Integración herramientas en sistemas de control de versiones 4d
33 Analisis de sistemas de control de versiones 3d
34 Propuesta de integración en distintas fases del SDLC 4d
35 Recopilar información sobre implementación 2d
36 Redacción de la memoria 3d
37 Entrega del PEC3
38 Fase final 30d
39 Revisión y complementación documentación 6d
40 Elaboración de conclusiones y recomendaciones 6d
41 Revisión estilo 5d
42 Revisión bibliográfica 3d
43 Redacción de memoria final 5d
44 Maquetación de memoria final 2d
45 Entrega del PEC4
46 Preparación vídeo presentación de la memoria 5d
47 Entrega vídeo presentación de la memoria

.1.6 Revisión del estado del arte

Como veíamos en la justificación de este trabajo, hoy en día podemos encontrar diversas soluciones y herramientas dirigidas a detectar fallos de programación o de diseño que pudieran ser explotados cuando la aplicación esté en marcha. Las metodologías que se analizan en este proyecto tratan de conseguir su objetivo a través de estrategias variadas, y sobre ellas se han creado soluciones informáticas de todo tipo, tanto propietarias[7] (ya sean gratuitas o de pago) como de código abierto.

Las herramientas que siguen la metodología SAST (Static Application Security Testing o Prueba de Seguridad de Aplicaciones Estáticas) son capaces de realizas pruebas de “caja blanca”. Es decir, este tipo de herramientas tienen acceso al código fuente, donde buscarán fallos de programación que pudieran ser explotados una vez la aplicación esté en marcha.

Los objetivos de la organización OWASP son ayudar a comprender las distintas vulnerabilidades de seguridad que existen en las aplicaciones web, enseñar a escribir código de forma más segura (con mecanismos y medidas pro-activas), y compartir métodos para securizar las aplicaciones. Las herramientas que proponen y proporcionan ayudan a equipos de desarrollo que quizá no cuenten con mucha experiencia en ciberseguridad a testear sus proyectos. La web de OWASP ofrece un listado muy amplio de herramientas SAST con distintos tipos de licencia[9]. De entre todas estas herramientas para analizar el código fuente de las aplicaciones, se estudiarán aquellas que tengan una licencia libre o sean gratuitas. Algunas herramientas se integran en los entornos de desarrollo para poder ver los problemas encontrados en el código fuente en tiempo real, haciéndolas más atractivas e intuitivas.

La metodología DAST (Dynamic Application Security Testing) engloba a aquellas herramientas orientadas a realizar pruebas de seguridad de aplicaciones dinámicas, también conocidas como pruebas de "caja negra". Este tipo de pruebas puede encontrar vulnerabilidades y debilidades de seguridad en una aplicación en funcionamiento, y se usan normalmente para testear aplicaciones web, pero no tienen acceso al código fuente ni a detalles de implementación o de la arquitectura de la aplicación. OWASP también ofrece un listado amplio de soluciones basadas en DAST[10] que serán analizadas en parte en la fase de investigación. Cabe destacar que OWASP también ofrece con licencia de código abierto su propio escáner de vulnerabilidades web, conocida como OWASP Zed Attack Procy (ZAP)[11].

Las herramientas que siguen la metodología DAST se conocen como escáneres de vulnerabilidades, y son capaces de encontrar vulnerabilidades potenciales que no son siempre explotables. Al ofrecer en ocasiones respuestas menos definitivas, suele ser necesario mayor análisis y testeo para valorar el riesgo real, por lo que muchas veces se combina una herramienta DAST con otra SAST. Afortunadamente, es habitual realizar el testing tanto en código fuente como en ejecución (SAST y DAST) para obtener una imagen

7 En el mundo del software libre se le conoce como «privativo» para indicar que quien use ese software tiene limitadas las posibilidades de uso, análisis, modificación o distribución mediante copia.

más global del sistema analizado, y el coste computacional de esta combinación no es grande ya que SAST requiere menos tiempo que DAST para analizar el código mientras ofrece resultados más fiables en las vulnerabilidades que encuentra.

Otra ventaja de SAST es que se puede implementar desde el inicio del desarrollo, mientras que DAST debe realizarse durante un proyecto ya en ejecución, con su código desarrollado y el entorno tanto instalado como configurado.

Para cuando no queramos combinar dos soluciones diferentes, podemos buscar una herramienta basada en IAST (*Interactive Application Security Testing* o Pruebas de Seguridad de Aplicaciones Interactivas) que realiza el análisis del código fuente en busca de vulnerabilidades de seguridad mientras la aplicación es ejecutada por una prueba automatizada o una persona.

Las herramientas IAST están orientadas a analizar aplicaciones web y Web APIs[8], pero desde OWASP solo mencionan una disponible de forma gratuita. La otra desventaja de IAST es que tiene un coste en recursos computacionales más alto, pero también es capaz de detectar mejor las vulnerabilidades ya que cuenta con un mejor contexto (y por tanto, ofrece menos falsos positivos)

En un contexto donde haya mucho software heredado (*legacy*) que se desarrolló en su día sin aplicar buenas prácticas de seguridad, una solución basada en SAST y DAST devolvería demasiados resultados a revisar y corregir, con muchos falsos positivos. IAST es más apropiado en este contexto porque tiene más datos de qué ocurre y dónde, y cómo afecta. Para conseguir ese objetivo, el agente IAST trabaja desde dentro de la aplicación lo que le permite controlar la ejecución y conocer en qué parte del sistema está la misma (código fuente, componentes externos, el backend…). En la siguiente tabla podemos ver un resumen de las diferentes características de estos tres tipos de analizadores de seguridad de aplicaciones:

SAST	**DAST**	**IAST**
Estático	*Dinámico*	*Interactivo*
Análisis proactivo	*Análisis proactivo*	*Análisis reactivo*
Aplicación parada	*Aplicación en ejecución*	*Aplicación en ejecución*
Acceso código	*Acceso externo*	*Acceso código*

Dentro del grupo de metodologías analizadas en este trabajo, tenemos por último las conocidas como SCA (Software Composition Analysys), que se implementan durante el desarrollo para mapear los componentes de código abierto que se utilizan en una aplicación. De estos componentes de código abierto externos se suelen conocer las vulnerabilidades de seguridad y los problemas de licencia de software que introducen al combinarse con otros paquetes de software, por lo que se pueden tomar decisiones y aplicar correcciones antes de comenzar a integrarlos en la solución web que se esté implementando.

8 Interfaz de programación de aplicaciones web

Las herramientas SCA analizan las dependencias que incluyen las aplicaciones para garantizar que no cuentan con vulnerabilidades conocidas, ya que se estima que hasta un 80% de los componentes que se incluyen en las aplicaciones web pueden tener vulnerabilidades[12]. SCA, por tanto, está más orientado a código de terceros, algo muy habitual y necesario en proyectos de software libre donde se combinan muchos componentes a la hora de construir una app. Desde OWASP nos ofrecen un chequeo de dependencias[13] basado en los diez riesgos de seguridad más importantes en aplicaciones web (OWASP Top 10[14]), y genera un perfil de riesgo que tendremos que analizar.

Además de OWASP, existen otras iniciativas abiertas donde se recomiendan herramientas y buenas prácticas de seguridad, como la iniciativa Build Security In (BSI[15]) de la Agencia de Ciberseguridad de los EEUU, o el programa SWAMP[16] cuyo objetivo es facilitar el testeo de calidad y seguridad de las aplicaciones de software.

.1.7 Análisis metodologías de desarrollo de software

Ciclo de vida del desarrollo de software

Conocemos como ciclo de vida del desarrollo de software (SDLC) al proceso en el que un proyecto de software se concibe, desarrolla y mantiene. La OWASP divide este proceso en cinco fases[17], aunque otras organizaciones prefieren plantearlo en un esquema de siete fases separando la planificación y el análisis como las dos primeras fases, y el despliegue y mantenimiento como las dos últimas.

Ilustración 1: Distintas fases que componen el SDLC (fuente: ViewNext)

Todo proyecto de software comienza con una fase de planificación de requerimientos que se acordará entre la parte que solicita la solución de software y el o la responsable de proyecto o representante del equipo/empresa que va a desarrollar la aplicación. En esta fase se determinan qué funcionalidades debería tener la aplicación y quedará plasmado en un documento o contrato ya que el resto del proceso se basará en las necesidades acordadas en esta primera fase. A fin de poder comprender mejor

cada fase del ciclo, y teniendo en cuenta que tanto OWASP como este trabajo de fin de máster se enfocan en soluciones web, vamos a contextualizar el ciclo de vida del desarrollo de software a una web sencilla que incluya elementos básicos (p.ej: una página de aterrizaje (*landing page),* la posibilidad de registrarse, ofrecer funciones de ingreso (*login*) o de salida del sistema).

Una vez recogidos los requerimientos del proyecto, los distintos equipos que se verán involucrados (desarrollo, *frontend/backend*, testers...), deben realizar un análisis de estos requerimientos con el objetivo de poder planificarlos mejor. Por ejemplo, para el requerimiento de registro, la aplicación web necesita un campo de recogida de nombre, otro para contraseña, un botón para aceptar los términos y condiciones del sitio web, el botón de envío, y la funcionalidad de poder guardar todos estos datos en una base de datos. El o la responsable de proyecto cogerá todas las funcionalidades identificadas en el análisis de requerimientos y los integrará en un sistema de gestión de proyectos.

En la fase de diseño se planifica el producto final en base a todos los requerimientos recogidos. En esta fase es donde se tienen que considerar las reglas de negocio, el diseño gráfico y la disposición de elementos de la interfaz (UI[9]), tecnologías, *frameworks* y lenguajes de programación requeridos para la implementación del producto, arquitectura del sistema, diseño de la base de datos, o la compatibilidad con los distintos dispositivos o navegadores web.

La fase de implementación es donde se programan y construyen los elementos y funcionalidades del producto. El equipo de programación y el de diseño de la interfaz web se coordinan para poder desarrollar la solución web, mientras que el equipo de testeo analiza el producto y construye los casos de prueba que servirán más adelante para determinar si cumplen con los requisitos. El equipo de testeo debe imaginar escenarios, casos y flujos de uso durante esta fase, a fin de poder encontrar errores de diseño al inicio de la fase implementación para así reducir el coste que pudiera requerir un rediseño del mismo.

La fase de pruebas se lleva a cabo una vez finalice la implementación de la solución web y todas las funcionalidades estén ya incorporadas a la solución. Esta fase siempre ha sido una de las más importantes en lo que respecta a la seguridad del sistema, ya que tradicionalmente aquí es donde se encontrarían errores en la implementación como por ejemplo, que cuando alguien pulse el botón de *logout*, no se estén borrando las *cookies* de sesión permitiendo que otra persona que utilice ese mismo equipo/navegador pueda acceder al perfil de la persona que supuestamente había abandonado su sesión.

Para poder detectar este y otro tipo de fallos que conllevarían un coste enorme (no tanto en corrección del *bug*, sino en daños y reputación si la web estuviera en marcha ya que las vulnerabilidades se correlacionan significativamente con una disminución del precio de las empresas[18]), el equipo de pruebas necesita que no solo esté todo el código programado, sino que todo el sistema (servidores, bases de datos…) esté integrado y en funcionamiento. Con el sistema completo en marcha pueden contrastar los casos de prueba planificados y validar el buen funcionamiento de cada requerimiento (es decir,

9 User Interface

que funcionan como se espera y que no tienen fallos o resultados inesperados). Los fallos de diseño y programación detectados se suelen integrar en un sistema de seguimiento de *bugs*, y el o la responsable de proyecto irá asignando los fallos al equipo de programación, ingeniería o diseño, a quienes les corresponda corregirlos. Los *bugs* suelen tener su propio ciclo de vida, desde que se detectan hasta que se consideran cerrados o corregidos.

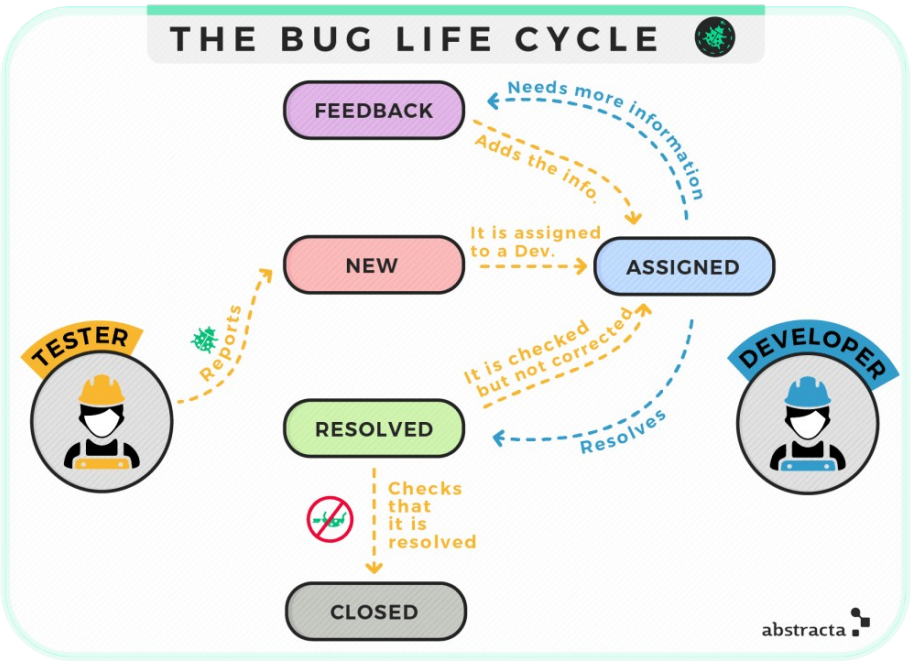


Ilustración 2: Ciclo de vida de errores (fuente: Abstracta)

Con el proyecto en un estado estable y funcional, se procede a la fase de despliegue. Tanto la fase de pruebas como esta de despliegue pueden tener unos requisitos de instalación similares. En muchos casos, se suelen dividir los entornos de despliegue en dos, con un entorno de desarrollo que como veíamos, debe incluir todos los sistemas y bases de datos requeridos por la aplicación, y otro de producción que podría requerir la instalación o el despliegue de las soluciones incorporadas a hardware más avanzado o potente ante la previsión de un uso más intensivo y de mayores exigencias que el realizado durante la fase de pruebas. En la fase de despliegue también se debe incorporar la solución web para que ésta funcione o bien de forma pública en Internet, o integrada en la Intranet de alguna otra organización.

Cuando la aplicación web está ya disponible para las personas que lo vayan a utilizar, nos encontramos en la fase final de uso y mantenimiento. Además de mantener los servidores y el entorno sobre el que corre la solución web, es necesario monitorizar la carga de tráfico de red o el consumo de recursos de memoria o cómputo provocado por el uso que realizan estas personas. En aplicaciones web, tradicionalmente esto se suele traducir en ampliaciones en la arquitectura del sistema para sostener mejor situaciones de sobrecarga, añadiendo más memoria, incluyendo balanceadores de carga, mejorando la respuesta de las consultas a la base de datos o ampliando el ancho de banda con el que se accede a la web.

Es importante considerar que aunque la aplicación esté terminada, quienes la usen puedan encontrar fallos del sistema que no hayan sido detectados por el equipo de pruebas, por lo que dependiendo del tipo de contrato de mantenimiento que se haya acordado, el equipo de desarrollo seguiría recibiendo estos reportes e incluirlos en el ciclo de vida de los *bugs*.

Modelo de desarrollo en cascada

Cuando las mencionadas fases del ciclo de vida del desarrollo de software transcurren de forma secuencial, nos encontramos ante un modelo de desarrollo conocido como "en cascada" o *waterfall*. Este modelo de gestión de proyectos de software es uno de los más sencillos de implementar, y por eso es habitual encontrarlo en desarrollos web que pueden determinar de forma definitiva todos los requerimientos de la aplicación en la primera fase.

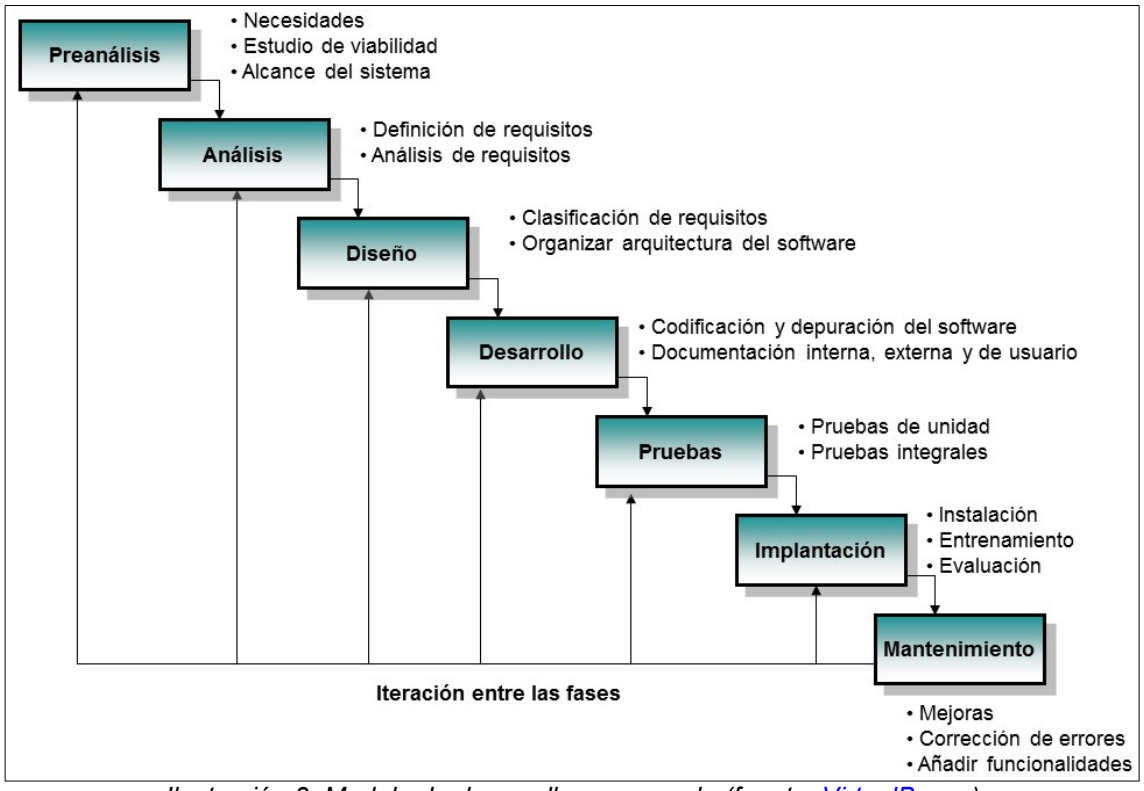


Ilustración 3: Modelo de desarrollo en cascada (fuente: VirtualBosco)

Este modelo requiere que cada fase se finalice antes de comenzar con la siguiente, lo que obliga a invertir mucho tiempo en la fase de recogida de requerimientos ya que una vez definidos, el modelo no va a permitir apenas cambios en las necesidades que cubrirá la aplicación web, las funcionalidades que incorporará o los problemas que trata de resolver.

Con el listado de requerimientos cerrado y analizado, se puede realizar el diseño lógico y físico del proyecto. En el diseño lógico se plasma una representación abstracta del flujo de datos, entradas y salidas del sistema; mientras que el diseño físico se incluyen aspectos que involucran al hardware necesario sobre el que se construye la solución de software. La fase de implementación no puede comenzar hasta que la fase de diseño no quede cerrada.

La fase de pruebas es la única que permite algo de flexibilidad, ya que al poder dividir las soluciones de software en distintos componentes, es posible llevar a cabo el testeo de algunos elementos programados e integrados en el sistema mientras que otros elementos siguen su implementación. Aunque el modelo de cascada permita dividir la implementación, el sistema en su totalidad debe ser verificado al finalizar completamente la fase de implementación.

Para el tipo de proyectos web que estamos planteando en este trabajo, las fases de despliegue y la de mantenimiento transcurrirían de forma secuencial.

El modelo de desarrollo en cascada es fácil de visualizar y comprender, y por tanto se hace sencillo de gestionar. Sin embargo, este modelo de desarrollo de software presenta varias desventajas para cierto tipo de proyectos ya que ofrece poca flexibilidad durante todo el ciclo y no es capaz de manejar con eficiencia los riesgos y cambios no previstos. En el caso de equipos pequeños, tampoco es conveniente este modelo para proyectos complejos o de largo plazo, ya que en estos, es difícil capturar al principio todos los requerimientos.

Modelo de desarrollo ágil

Para solventar los problemas de flexibilidad que muestra el modelo de desarrollo en cascada, se plantean los modelos conocidos como ágiles que plantean un proceso compuesto por fases iterativas.

Recordemos que el objetivo de este trabajo es seleccionar un conjunto de herramientas SAST, DAST, IAST y/o SCA planteadas por la OWASP y encontrar la forma de integrarlas dentro del ciclo de desarrollo y despliegue de una aplicación. Como iremos viendo a partir del apartado 2.3 que analiza las distintas soluciones posibles, estas herramientas plantean escenarios más disruptivos que los encontrados en la tradicional fase de pruebas del modelo de cascada, ya que pueden integrarse en distintos momentos del ciclo de desarrollo (en la fase de diseño, en la implementación, en la fase de pruebas o en las fases de despliegue y mantenimiento), por lo que es posible que convenga integrarlas en un modelo de desarrollo que permita una perspectiva más iterativa tanto de las fases como de la evolución del propio proyecto.

Ilustración 4: Modelo de desarrollo tradicional VS ágil (fuente: IEBS school)

La metodología ágil dentro de un desarrollo de software busca dividir el proyecto en versiones limitadas que se conocen como “producto viable mínimo” o MVP[19]. El MVP se refiere a la versión más pequeña y simple de lo que el proyecto web trata de ofrecer, y el objetivo es tener lanzada cuanto antes esa versión mínima pero funcional para poder obtener el retorno sobre dudas, fallos, sugerencias de mejora o ver qué funciona mejor o peor entre las funcionalidades implementadas.

Esta filosofía suele ser seguida por multitud de equipos de desarrollo con gran pasión y alto grado de implicación, hasta el punto de que existe la posibilidad de mostrar lealtad firmando[20] lo que se conoce como el “Manifiesto para el desarrollo ágil de software”[21]:

> *Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:*
>
> ***Individuos e interacciones*** *sobre procesos y herramientas*
> ***Software funcionando*** *sobre documentación extensiva*
> ***Colaboración con el cliente*** *sobre negociación contractual*
> ***Respuesta ante el cambio*** *sobre seguir un plan*
>
> *Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.*

El énfasis que le pone el manifiesto y por ende, las metodologías ágiles, a valorar las respuestas ante cambios frente a lo planificado puede facilitar la incorporación de herramientas disruptivas a todo el proceso de desarrollo. Recordemos que algunas herramientas DAST tienden a dar muchos falsos positivos, por lo que una aproximación ágil e iterativa puede hacer más manejable la incorporación de ese *feedback* al proceso de desarrollo[10].

También encaja con nuestro escenario de enfocarnos en desarrollos web que utilizan componentes de código abierto y realizados por equipos pequeños, ya que normalmente los proyectos desarrollados siguiendo metodologías ágiles suelen componerse de equipos de tres a nueve personas. De acuerdo a Scrum[22] (un marco de trabajo para el desarrollo ágil de software), este tipo de metodologías están pensadas para un tamaño de equipo lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar un trabajo significativo.

Las metodologías ágiles permiten construir cierta tolerancia a los errores que se cometen durante el desarrollo del software, pudiendo aprender de ellos, pero siempre con el objetivo de que después de cada iteración y a medida que nos acercamos a la versión final/completa, se cometan menos errores y se solucionen más rápido.

Para conseguir este marco de trabajo, las metodologías ágiles se basan en tres características principales: los *sprints*, el *scrum master* y los *stand ups*.

10 Analizaremos varias estrategias para abordar ese problema en el capítulo 3.1.1

Un sprint[23] es una iteración de todo un ciclo dentro de un desarrollo ágil, y normalmente transcurre en un breve periodo de tiempo (p.ej, dos o tres semanas). En un breve sprint podemos ver todas las fases que vemos en el tradicional ciclo de vida del desarrollo de software, pero cada fase tiene una duración más corta y el retorno de quienes usen y testeen la app, y la consecuente respuesta del equipo de desarrollo y diseño, es cada vez más rápida y ajustada. En cada sprint se atiende un número de requisitos de los cuales se indica también cuándo una tarea se considera finalizada, cuánto debería durar su implementación y si depende de otras tareas o características para ser implementado. El seguimiento de los requisitos se hace a través de *tickets*, y las tareas que representan se gestionan a través de sistemas de seguimiento de incidencias como JIRA[24] o Pivotal Tracker[25].

La planificación de los *sprints* tiene en cuenta que a veces una iteración puede durar más o menos de lo previsto, y al final de cada *sprint* se determinan los requerimientos que se van a implementar en la siguiente iteración. Esta fase final del *sprint* se conoce como retrospectiva, y es el momento que se aprovecha para analizar qué fue bien durante la iteración y qué se puede mejorar.

El *scrum master* es una persona que aunque asume algunas tareas tradicionales de responsable de proyecto, no tiene porque tener formación tecnológica. Su labor principal es ayudar al equipo a terminar las tareas pendientes en cada sprint asegurando que todas las personas involucradas tienen lo que necesitan para llevar a cabo sus responsabilidades, y atendiendo también a otro tipo de necesidades (motivacionales, de convivencia…).

Finalmente, en proyectos de metodología ágil, cada día se hace una breve reunión en la que no hace falta sentarse (de ahí su nombre en inglés de *stand up[26]*) para asegurar tanto la brevedad de la misma, como la ausencia de distracciones al ser más difícil estar pendiente del ordenador o del móvil durante la reunión. En esta breve actualización de estado se informa al resto sobre lo que se ha estado realizando, así como hacer saber de algún bloqueo que se hayan encontrado o si están pendientes de la tarea de otra persona.

La metodología ágil se considera óptima para aquellos proyectos que requieren ser publicados rápido y necesitan muchos cambios pequeños. La metodología en cascada prioriza la calidad sobre el tiempo de desarrollo, y exige que la solución funcione tal como se espera desde el principio. Por tanto, en algunos escenarios es necesario mezclar ambas metodologías, cascada y ágil, a fin de encontrar una metodología más adaptada. Proyectos tradicionales de gran envergadura (por ejemplo, una aplicación web para un banco o un hospital), suelen seguir la metodología de desarrollo en cascada de forma más estricta, mientras que un proyecto sencillo o minimalista (p.ej, una app móvil o una página web meramente informativa) se ajusta mejor a la metodología ágil. La metodología de desarrollo ágil encaja como anillo al dedo con la filosofía principal de los proyectos de código abierto de “publicar temprano, publicar a menudo”[27], por lo que en el tercer capítulo veremos cómo integrar en el ciclo de vida del desarrollo de software siguiendo la metodología ágil algunas de las herramientas que se analizan en este segundo capitulo.

.1.8 Recursos y presupuesto del proyecto

Para la realización de este trabajo será necesario el uso de un ordenador personal en el que además de estudiar las características técnicas de las distintas herramientas que se analizarán, se instalarán y probarán algunas de ellas en aquellos casos que el análisis así lo requiera. Para estos casos, se utilizarán versiones gratuitas o software de código abierto. El coste de este equipo está estimado en unos **600€**

Además de la infraestructura necesaria para la conexión a Internet, con un coste de unos **39€/mes**, se implementará un sistema de copias de seguridad automatizado a fin de poder respaldar cada día la documentación y materiales analizados durante este trabajo. Las copias se realizarán en un dispositivo NAS[11] que tiene un coste de **200€**.

El presupuesto total del proyecto para los cinco meses en los que se desarrolla es de **995€**. No hay coste adicional por software ya que se instalarán y analizarán en el ordenador personal tan solo las herramientas disponibles de forma libre o gratuita, y el sistema operativo utilizado en el ordenador personal será GNU/Linux, a través de la distribución Ubuntu.

.1.9 Análisis de riesgos

En este apartado indicaremos brevemente varios de los riesgos que pueden surgir durante el desarrollo de este proyecto y que pudieran causar cambios en la planificación, el enfoque o en los objetivos marcados:

- Que el número de herramientas disponibles para cada metodología sea demasiado extenso como para poder analizar con detenimiento una buena muestra de la misma. (RIESGO MEDIO)
    - *Se acotará el grupo de candidatos en base a unos criterios: qué la herramienta sea de código abierto, funcione en sistemas Windows o GNU/Linux, soporte lenguajes comunes en el desarrollo de aplicaciones web y esté relativamente actualizada.*
- Debido a su antigüedad, que algunas herramientas SAST y DAST no se adapten a los nuevos escenarios que muestran las aplicaciones web o móviles. (RIESGO ALTO)
    - *En este caso, IAST se adapta mejor pero hay menos herramientas disponibles de forma gratuita o libre, por lo tanto, se descartarán aquellas herramientas SAST y DAST que no hayan actualizado al menos los perfiles y plugins para atender vulnerabilidades y escenarios recientes.*
- Que la complejidad y esfuerzo a la hora de integrar y configurar la automatización de las herramientas sea demasiado grande para el objetivo marcado en el trabajo. (RIESGO BAJO)

11 *Network-attached storage* o almacenamiento conectado en red

- *Se podrá analizar la herramienta en base a la documentación, experiencias y revisiones realizados por terceros.*

- Que las herramientas automatizadas creen una falsa sensación de seguridad, y promueva al equipo de desarrollo a no implementar de forma pro-activa prácticas de programación segura pensando que sus errores se van a detectar por las herramientas. (RIESGO MEDIO)
  - *Se planteará una guía de buenas prácticas para equilibrar los resultados de las auditorías con la necesidad de mantener código seguro. Los resultados de las auditorías deben servir de aprendizaje también.*
- Que la propuesta de herramientas y procedimientos acabe generando un *backlog*[12] de vulnerabilidades inmanejable (p.ej, con demasiados falsos positivos). (RIESGO ALTO)
  - *Se seleccionarán aquellas herramientas que ayuden a reducir el número de falsos positivos, y se propondrá la integración de las soluciones SAST/DAST/IAST o SCA en ciclos de desarrollo más cortos y ágiles a fin de reducir el backlog de vulnerabilidades y facilitar su tratamiento.*

12 La acumulación de tareas o trabajos pendientes

.2. Fase de investigación

.2.1 Prácticas de programación seguras

Para cumplir con su objetivo de ayudar a los sitios web y a personas expertas en seguridad a proteger las aplicaciones web de los ciberataques, la OWASP ofrece una serie de recomendaciones relacionadas con la programación segura[28]. Para ello proponen una lista de verificación (*checklist*) que puede integrarse en el ciclo de desarrollo de distintos tipos de proyectos de software, y son lo suficientemente genéricos como para tenerlos en cuenta independientemente de los lenguajes de programación usados en el proyecto.

Dado que las herramientas SAST, DAST, IAST o SCA arrojan resultados que requieren cambios en el código fuente, es importante conocer buenas prácticas de programación segura para poder implementar las correcciones necesarias, así como saber identificar qué parte del código y cómo se debe modificar para evitar las vulnerabilidades y *exploits* detectados en algunas herramientas.

La programación segura es condición *sine qua non* para lograr que el software mantenga sus principales características de seguridad entorno a la confidencialidad, integridad y disponibilidad de los datos. La lista de verificación que propone OWASP está pensada para reducir las vulnerabilidades más frecuentes en proyectos de software, y tienen estrecha relación con los diez riesgos de seguridad más importantes que enumeran cada año para las aplicaciones web (OWASP Top 10).

OWASP Top 10 – 2013		OWASP Top 10 – 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Ilustración 5: Comparativa de riesgos principales para webs en 2013 vs 2017 (fuente: OWASP)

Seguir las prácticas de programación segura planteadas por la OWASP sirve también para comprender e incorporar los escenarios de abuso comunes (interacciones que una persona puede llevar a cabo y que son perjudiciales para el sistema) al testeo que debe realizarse tanto al código como al sistema en su conjunto. Estos casos de abuso no solo ocurren por errores introducidos durante la fase de implementación, sino que también pueden ser fruto de un

mal análisis de requerimientos, un error en el diseño de la arquitectura de software, o por la falta de una política de mantenimiento y actualización adecuada de los distintos componentes que comprenden el proyecto.

Buenas prácticas de programación segura:

Cada uno de los catorce grupos aglutina varios pasos que se debería tomar para verificar que se han incorporado e implementado las recomendaciones a nuestro proyecto de software. En este apartado, mencionaremos brevemente algunos de los aspectos que habría que tener en cuenta a la hora de desarrollar un sitio web securizado.

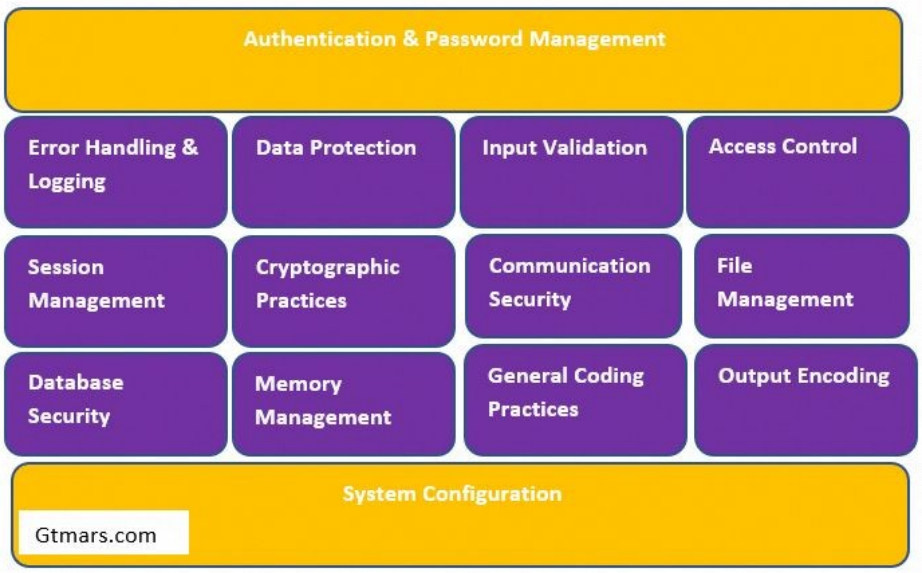


*Ilustración 6: Esquema buenas prácticas de programación segura de OWASP (*Fuente: Hackermoon)

1. **Validación de entradas**

La validación de datos debe hacerse tanto a nivel de cliente (cuando recogemos los datos desde el navegador), como a nivel de servidor. Es necesario considerar y tratar apropiadamente los datos que vengan de fuentes no confiables (p.ej, de bases de datos externas) y establecer una codificación de datos común (p.ej, UTF-8[13]).

Para protegerse de algunos ataques, es necesario validar escenarios donde los datos de entrada sean mayores o de tipo diferente a lo esperado, y para combatir la construcción de instrucciones maliciosas basadas en caracteres peligrosos se pueden integrar herramientas y API de seguridad para ayudar en la validación.

2. **Codificación de salidas**

Se recomienda codificar todos los caracteres salvo que sean reconocidos como seguros por el interprete de destino, y esta codificación debe hacerse en un entorno seguro (p.ej: en el servidor). Para securizar datos potencialmente

13 8-bit Unicode Transformation Format, formato de codificación de caracteres

peligrosos y evitar inyecciones de código[29], se utilizan técnicas para sanitizarlos, eliminando, reemplazando, codificando o invocando una interpretación alternativa de los mismos (*i.e.* escapando).

3. Administración de autenticación y contraseñas

La exposición de datos sensibles sigue siendo uno de los errores más explotados[30] de las páginas web, por lo que para evitar que existan errores en los controles de acceso[31], todos los recursos (exceptuando las páginas web públicas) deben requerir autenticación. Estos controles se deben llevar a cabo en el servidor y estar basados en estándares probados.

Se deben utilizar funciones *hash* con sal[32] a la hora de almacenar las contraseñas en el servidor, y si se envía una nueva contraseña generada, hay que hacerlo por un canal cifrado exigiendo el cambio a una contraseña segura en base a una política adecuada. La inhabilitación de contraseñas, la solicitud de su cambio o las opciones de recuperarla deben implementarse siguiendo protocolos y políticas de seguridad apropiadas. Ante errores de autenticación hay que evitar dar pistas de qué han introducido incorrectamente. Las credenciales necesarias para acceder a servicios externos se cifran y guardan en servidores de confianza.

También es recomendable tener siempre activado un sistema de monitorización[33] para identificar ataques y poder activar medidas de contención (bloqueo de cuentas, envío de avisos…), así como tener un sistema de autenticación multi-factor para añadir una capa más de seguridad ante el robo de una contraseña o un ataque de fuerza bruta.

4. Administración de sesiones

El sistema solo debe aceptar los controles del servidor o del *framework* para la administración de sesiones. Los identificadores se deben crear en un servidor de confianza y deben ser lo suficientemente aleatorios como para no predecir el patrón.

Además de garantizar de que al salir de la sesión se borren todos los identificadores, debe existir la opción de salir en todas las páginas que son solo accesibles bajo autenticación. También se podrá salir de la sesión al pasar un tiempo de inactividad preestablecido, y en algunos casos se debe bloquear que una misma persona se autentique al mismo tiempo desde dos lugares distintos.

El Reglamento General de Protección de Datos[34] exige la inclusión del cifrado en la conexión cuando se transmiten datos personales, por lo que se recomienda activar siempre el protocolo HTTPS. Para prevenir ataques de falsificación de petición en sitios cruzados (CSRF[35]) se pueden utilizar de forma complementaria *tokens* o parámetros de sesión durante el manejo de la sesión.

5. Control de Acceso

Se debe requerir siempre controles de acceso en cada solicitud que se haga al sistema, incluyendo los que se hagan desde parte del código que se ejecuta del lado de cliente (en JavaScript u otro lenguaje). Las URL, objetos, funciones, datos, ficheros y otros recursos disponibles solo para personas autorizadas deben estar restringidas en todos los demás casos.

Para evitar abusos, es conveniente limitar el número de transacciones que pueda realizar una persona usuaria común. Así mismo, se recomienda re-validar la autorización durante sesiones de autenticación largas ya que durante ese tiempo los privilegios de esa cuenta han podido cambiar. Las cuentas usadas para integrar distintos componentes (p.ej, para conectar con la base de datos), deben tener el mínimo privilegio.

6. Prácticas Criptográficas

Las operaciones de cifrado se hacen siempre en el lado del servidor, y se protegen de forma especial todas las llaves maestras. El manejo de claves de cifrado se debe basar en una política establecida, y para evitar intuir patrones, la generación de elementos aleatorios debe hacerse a través de componentes cuya seguridad y efectividad esté verificada.

7. Manejo de errores y Logs

Al mostrar un estado de error no se debe ofrecer datos sensibles, ni información de depuración, ni detalles de la implementación del sistema. Después de cada error se debe liberar la memoria que ya no es requerida.

El registro de errores y casos de éxito debe realizarse en un servidor de confianza, y serán accesibles solo por personal autorizado. En los *logs* no se debe guardar información sensible (p.ej, una contraseña).

8. Protección de datos

Para cada operación se debe implementar el mínimo privilegio requerido para ofrecer la funcionalidad esperada. Los archivos temporales (p.ej, la caché) que incluyan datos sensibles además de protegidos (cifrados y/o con control de acceso), se deben eliminar en el momento que dejen de ser necesarios. Contraseñas, datos de conexión y demás información sensible se deben almacenar también cifrados.

En proyectos de código abierto o cuando el cliente tenga acceso al código fuente privado, no se deben incluir comentarios que revelen información sensible.

9. Seguridad en las comunicaciones

Toda la comunicación debería estar cifrada, asegurando que los certificados TLS[14] son válidos y fiables. No conviene rebajar la comunicación a HTTP no seguro cuando falle la conexión TLS, y nunca debe permitirse acceder a recursos que requieran autenticación sin tener el protocolo HTTPS activado.

10. Configuración de los sistemas

Todo componente utilizado en la construcción de la solución web debe estar actualizado o con los parches de seguridad pertinentes. El sistema no debe tener activos servicios y funciones no requeridas en la aplicación web, y los servicios activos deben tener el mínimo privilegio posible. No hay que permitir que se puedan listar los directorios, y tampoco incluir su estructura en el archivo *robots.txt*[15]*.*

Los entornos de desarrollo y de producción deben estar en localizaciones diferentes y aisladas entre sí. Para registrar y manejar los cambios en el código de la aplicación, se recomienda utilizar un sistema de control de versiones (*Git, Subversion*…).

11. Seguridad de Base de Datos

Para reducir los ataques de inyección SQL se pueden usar consultas parametrizadas, validar las entradas, codificar las salidas o realizar una gestión apropiada de privilegios de acceso a la BBDD desde las aplicaciones de cliente. No se debe incluir nunca en texto plano en la aplicación los valores de autenticación y configuración para conectar a la base de datos (es mejor que estén cifrados y separados en un sistema fiable).

12. Manejo de Archivos

Toda transferencia de archivo al servidor debe requerir autenticación previa, y solo se deben aceptar ficheros requeridos por la operativa de negocio, con la correspondiente validación para que no se envíen archivos con la extensión cambiada; adicionalmente, después de descargar se puede realizar un análisis en busca de *malware*. Estos archivos se guardan en un contexto separado de la web (una base de datos o repositorio) eliminando (si los tuvieran) los permisos de ejecución .

Para no dar pistas de la estructura interna, se debe evitar mostrar la ruta e incluso nombres de los archivos ofrecidos, usando índices e identificadores en su lugar.

13. Manejo de Memoria

Para evitar la explotación de un desbordamiento de búfer[16] hay que revisar la longitud de los mismos, y utilizar controles en la entrada y salida de datos.

14 *Transport Layer Security*, seguridad de la capa de transporte

15 Archivo que indica a rastreadores de motores de búsqueda qué páginas/archivos pueden o no solicitar de su sitio web

Conviene liberar recursos de forma directa (sin confiar solo en el recolector de basura[17]) y evitar el uso de primitivas con vulnerabilidades conocidas.

14. Practicas Generales para la Codificación

Utilizar siempre código fiable y probado, así como las APIs oficiales o de demostrado reconocimiento, evitando aquellas que invoquen una línea de comandos (*shell*) para ejecutar otros comandos en el sistema operativo. Para verificar la integridad de bibliotecas, ejecutables y otros archivos utilizados se pueden usar funciones *hash* y comparar con los valores oficiales de las versiones utilizadas.

Si una aplicación requiere permisos especiales, se elevarán solo para ese momento. Las aplicaciones de terceros que se utilicen en el sistema deberán ser revisadas para validar su funcionamiento seguro (por ejemplo, con una herramienta SCA). Por último, las condiciones de secuencia[18] se pueden evitar utilizando semáforos.

.2.2 Herramientas para Pruebas de Seguridad de Aplicaciones Estáticas

.2.2.1 Análisis herramientas SAST

Las herramientas para realizar pruebas de seguridad de aplicaciones estáticas (SAST - Static Application Security Testing) son también conocidas como herramientas de análisis del código fuente[36]. Al trabajar sobre el código fuente, muchas de estas herramientas se integran directamente en los entornos de desarrollo integrados (IDE) como Eclipe[37] o NetBeans[38], dos suites distribuidas con licencias de código abierto[39][40]. Esta integración permite detectar y corregir casi de forma inmediata durante el proceso de desarrollo de la aplicación web, reduciendo la probabilidad de acumular errores o provocar problemas en cadena al no atender la falla de seguridad desde su primera incursión en el código fuente.

Del listado de herramientas propuesto por la OWASP, se han seleccionado y analizado aquellas que además de ser de código abierto, realicen el análisis de código fuente en lenguajes frecuentemente utilizados en desarrollos web que utilicen componentes de software libre. Este tipo de proyectos suelen basarse en lenguajes de programación como Python, PHP o Javascript, por lo que no se analizarán herramientas de análisis del código fuente que solo soporten lenguajes como C/C++ o .NET.

16 Cuando datos escritos en un búfer también corrompen valores de datos en direcciones de memoria adyacentes al búfer de destino debido a una comprobación de límites insuficiente (buffer overflow)

17 Mecanismo automatizado para gestionar la memoria (reservar/liberar/compactar...)

18 Cuando dos o más procesos acceden a datos compartidos y tratan de modificarlos al mismo tiempo (race condition)

De entre los puntos fuertes de las herramientas SAST podemos destacar la mencionada integración con los IDE, así como la posibilidad de realizar análisis frecuentes sobre todo el código fuente para poder estar informado en el día a día de los posibles errores de seguridad. Normalmente, las herramientas SAST ofrecen *feedback* útil y rápido de interpretar, ya que resaltan directamente la línea de código en el IDE o indican en qué fichero y línea se encuentra el error. Son especialmente útiles para encontrar variables donde puedan explotarse desbordamientos de búfer o realizarse inyecciones SQL.

Las herramientas SAST también presentan una serie de debilidades a considerar. Existen multitud de vulnerabilidades de seguridad que no son fáciles de detectar con herramientas automatizadas (p.ej, aquellas que tengan que ver con problemas de autenticación, control de acceso, o un uso criptográfico débil), por lo que siempre hay que considerar que estas herramientas detectan un porcentaje de los errores, y es más importante seguir prácticas de programación segura en todo momento.

También podemos recibir numerosos falsos positivos que no haría falta corregir, y no siempre podrán analizar el código si este no puede ser compilado, lo que obliga a usar estas herramientas cuando el proyecto está en un estado más completo y/o funcional. Este problema se vería mitigado en un escenario donde se esté desarrollando el proyecto web siguiendo una metodología ágil, ya que cada iteración corta da como resultado un proyecto completamente utilizable, y por tanto, las herramientas SAST tendrían posibilidad de analizar el código fuente junto con todas las dependencias (librerías, *frameworks*…) cubiertas.

.2.2.2 Selección herramientas SAST

Agnitio Security Code Review Tool

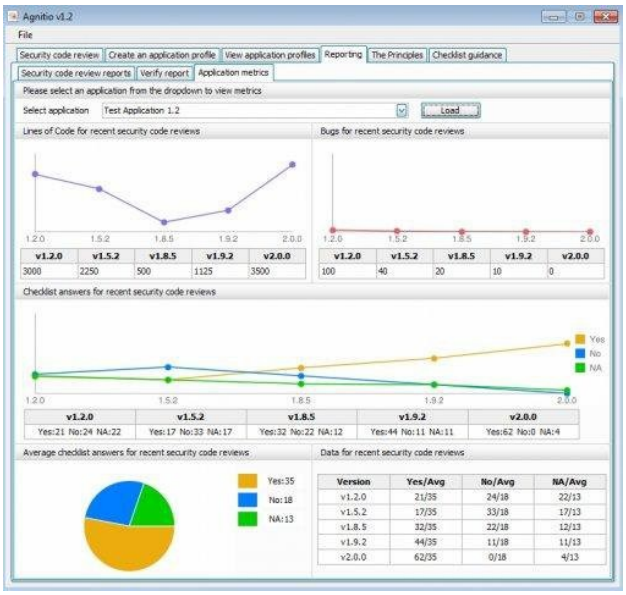


Ilustración 7: Interfaz principal de la aplicación Agnitio (fuente: Sourceforge)

- **Sitio web**: https://sourceforge.net/projects/agnitiotool/files/
- **Plataforma**: Windows
- **Fecha última versión**: 24-10-2011
- **Licencia:** GNU General Public License version 3.0 (GPLv3)[41]
- **Lenguajes de programación soportados:**
  - ASP, ASP.NET, C\#, Java, Javascript, Perl, PHP, Python, Ruby, VB.NET, XML
- **Puntos fuertes:**
  - Ofrece un *checklist* para ir revisando distintos aspectos a analizar, siguiendo una filosofía similar a las recomendaciones de practicas de programación seguras de la OWASP.
  - Ayuda a que el proceso de revisión sea fácil y repetible.
  - Muestra el código fuente resaltando el error, y propone casos de prueba para ellos.
  - A pesar de no poder automatizar el análisis, la herramienta permite realizar análisis manual de forma consistente y repetible.
  - La inclusión de métricas permite medir las mejoras, haciendo que sea adecuado incorporar la revisión manual a cada sprint dentro de una metodología de desarrollo ágil.
  - Sencillo de utilizar,
- **Puntos débiles:**
  - No se integra en IDE, solo hace análisis de código de manera manual.
  - Sin mantenimiento, lleva diez años sin estar actualizado
  - No tiene versión para GNU/Linux disponible
- **Informes:**
  - Produce registros de auditoría y aplica controles de integridad
  - En una misma herramienta se ofrecen los informes con los resultados junto con las métricas.

Bandit

Ilustración 8: Resultado análisis de Bandit por línea de comandos (fuente: Webner)

- **Sitio web**: https://bandit.readthedocs.io
- **Plataforma**: Diseñado para GNU/Linux
- **Fecha última versión**: 13-12-2020
- **Licencia:** Apache-2.0
- **Lenguajes de programación soportados:**
  - Python (hasta versión 3.9)
- **Puntos fuertes:**
  - Instalación sencilla a través de gestores de paquetes de Python como PIP[42]
  - Es configurable para cubrir distintas necesidades (desde proyectos de un solo desarrollador a procesos de automatización y despliegue continuo)[43]
  - La simplicidad del comando y los rápidos resultados por fichero lo hacen apropiado para ser usado no solo durante el desarrollo, sino para analizar proyectos/componentes que se quieran integrar.
  - Permite analizar en base a perfiles de seguridad (p.ej: buscar posibles ataques de inyección de instrucciones *shell*)[44]
  - Se integra con sistemas de control de versiones como Git
- **Puntos débiles:**
  - No se integra en entornos de desarrollo, y la herramienta está solo disponible para la línea de comandos.
  - Solo disponible para lenguaje de programación Python.
- **Informes:**
  - Después de procesar cada fichero Python, muestra en un informe los resultados ordenados por prioridad y confianza
  - Los informes pueden ser exportados en formato CSV, HTML o JSON

SearchDiggity 3.1

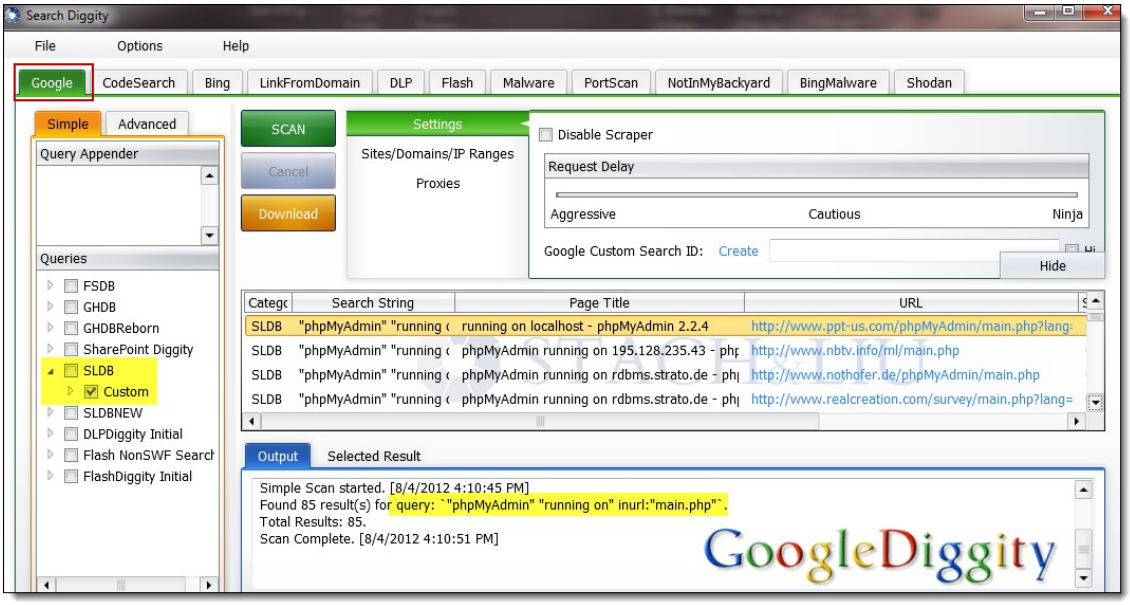


Ilustración 9: Resultado del análisis de herramienta GoogleDiggity (Fuente: BishopFox)

- **Sitio web**: https://resources.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/
- **Plataforma**: Windows
- **Fecha última versión**: 31-3-2014
- **Licencia:** Freeware.
- **Lenguajes de programación soportados:**
  - Puede realizar análisis de seguridad del código fuente de muchos proyectos de código abierto alojados en varios servicios.
- **Puntos fuertes:**
  - Es parte de lo que se conoce como Open Source Intelligence (OSINT), una forma de realizar investigaciones utilizando información de fuentes abiertas.
  - Utiliza Google Code Search para identificar vulnerabilidades en proyectos de código abierto alojados en Google Code, MS CodePlex, SourceForge, Github y otros.
  - Basta con indicar la URL del repositorio para realizar en análisis
  - Identifica problemas de inyección SQL, cross-site scripting (XSS), contraseñas codificadas…
  - Integra varias herramientas del proyecto Google Hacking Diggity[45], diseñados para buscar distintos tipos de vulnerabilidades
- **Puntos débiles:**
  - No disponible para sistemas GNU/Linux
  - No puede analizar código que no esté publicado de forma abierta en los servicios y plataformas de desarrollo que tiene integrados.
  - Encuentra solo vulnerabilidades publicadas
- **Informes:**
  - Los resultados se muestran en la aplicación listados y ordenados por categoría y sub-categoría.
  - Al seleccionar cada ítem se puede ver la parte de código que contiene la vulnerabilidad.
  - Permite exportar los resultados en formato XLS y HTML.

Horusec

Ilustración 10: Informe visual del panel de gestión de Horusec (Fuente: Horusec)

- **Sitio web**: https://horusec.io
- **Plataforma**: GNU/Linux, Windows
- **Fecha última versión**: 23-3-2021
- **Licencia:** Apache-2.0
- **Lenguajes de programación soportados:**
  - C#, Java, Kotlin, Python, Ruby, Golang, Terraform, Javascript, Typescript, Kubernetes, PHP, C, HTML, JSON, Dart, Elixir, Shell.[46]
- **Puntos fuertes:**
  - La herramienta está en activo desarrollo y tiene actualizaciones frecuentes
  - Se integra con sistemas de control de versiones como Git
  - Busca fugas y fallos de seguridad tanto en los ficheros de un proyecto, como en el historial de Git.
  - Adaptable a equipos pequeños y grandes.
  - Se integra a distintos entornos de desarrollo a través de extensiones.
- **Puntos débiles:**
  - Gestión de la herramienta a través de la línea de comandos, sin interfaz gráfica para realizar los análisis.
  - Todavía no realiza análisis semántico del código, pero esperan tenerlo implementado para finales de 2021.
- **Informes:**
  - Los informes se ofrecen a través de una aplicación web.
  - Los resultados se pueden clasificar de distintas formas, y la herramienta web ofrece numerosas métricas para facilitar el seguimiento.

Automated Software Security Toolkit

Ilustración 11: Informe en formato HTML del analisis de ASST (Fuente: OWASP)

- **Sitio web**: https://owasp.org/ASST
- **Plataforma**: GNU/Linux, MacOSX y Windows
- **Fecha última versión**: 31-10-2020
- **Licencia:** MIT License[47]
- **Lenguajes de programación soportados:**
  - PHP y MySQL
- **Puntos fuertes:**
  - Enfocado en el escaneo de vulnerabilidades web basados en el Top 10 de OWASP.
  - Fácil de instalar [48] sobre un paquete XAMPP junto con el entorno de tiempo de ejecución de JavaScript Node.js.
  - Analiza también si las versión de PHP y MySQL en uso son vulnerables.
- **Puntos débiles:**
  - Solo disponible como herramienta para la línea de comandos.
  - De momento tan solo soporta los lenguajes PHP y MySQL, pero permite extensiones que en un futuro añadan soporte a otros lenguajes como Java, C# o Python.
- **Informes:**
  - Después de escanear un proyecto, permite clickar cualquier línea del informe con el error detectado para obtener información adicional sobre por qué es vulnerable y cómo corregirlo.
  - Los informes se muestran en formato HTML e incluyen los ficheros PDF que explican los posibles ataques y los consiguientes mecanismos de protección.

.2.3 Herramientas para Pruebas de Seguridad de Aplicaciones Dinámicas

.2.3.1 Análisis herramientas DAST

La herramientas DAST (Dynamic Application Security Testing) se conocen también como escáneres de vulnerabilidad de aplicaciones web. A diferencia de las herramientas SAST, que son capaces de analizar el código fuente de un proyecto, DAST aborda el análisis desde fuera, usando una perspectiva de "caja negra".

En las aplicaciones web, las vulnerabilidades más frecuentes suelen ser las inyecciones SQL o de otros comandos, las configuraciones inseguras del servidor, o los ataques de tipo XSS[19]. Una herramienta DAST intenta replicar de forma automatizada el trabajo que haría una persona con formación en ciber-seguridad que realiza un prueba manual de penetración (*pentesting*) para buscar vulnerabilidades en la aplicación.

Toda herramienta DAST se compone de al menos dos elementos: una araña rastreadora[49] que recorre toda la página web para capturar la mayor cantidad de recursos (URL[20]) posibles, y un motor de detección que envía peticiones a cada URL incluyendo *payloads*[21] de ataque. Dependiendo del tipo de aplicación web y las tecnologías incorporadas en el mismo, las herramientas DAST permiten configurar el tipo de *payloads* de ataque que queremos probar. De esta forma se puede reducir el tiempo de cada análisis al enfocarnos solo en las vulnerabilidades más probables.

Existen multitud de herramientas DAST disponibles, tanto comerciales como de código abierto, y en la web de OWASP se comparten varias. La propia OWASP tiene una herramienta[50] para analizar el rendimiento de estas soluciones DAST, y comparte varios resultados.

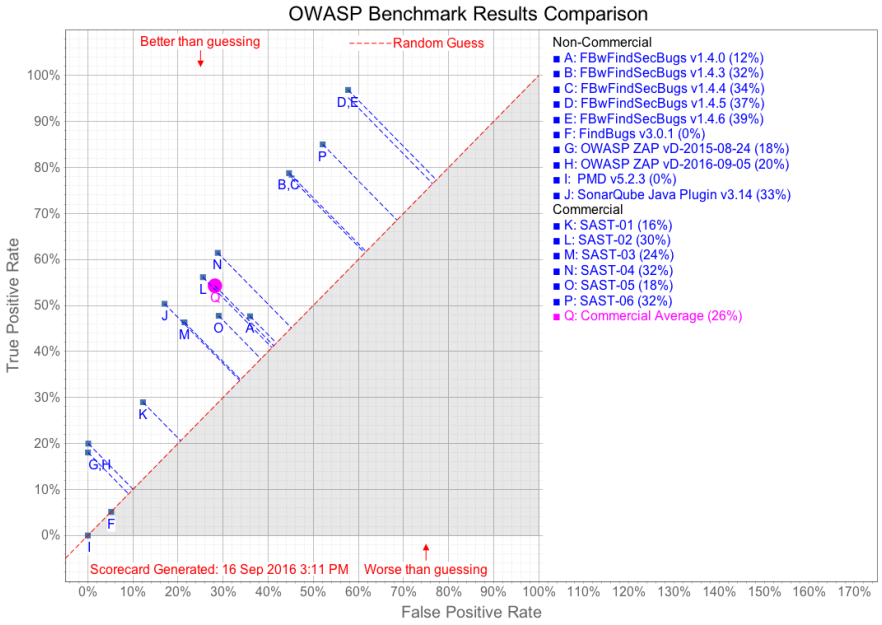


Ilustración 12: Resultados de rendimiento de varias herramientas DAST (Fuente: OWASP)

19 Cross-site scripting o secuencia de comandos en sitios cruzados
20 Uniform Resource Locator, localizador de recursos uniforme
21 El componente del ataque que causa daño al sistema

Existe también un proyecto llamado WAVSEP (*Web Application Vulnerability Scanner Evaluation Project*) que analiza cuatro herramientas DAST de código abierto[51], dos de las cuales aparecen en el listado de OWASP.

Frente a los análisis estáticos, que tienen que estar diseñados para lenguajes de programación concretos, DAST tiene la ventaja de que escanea una aplicación web independientemente de los lenguajes y *frameworks* sobre el que se haya construido. Es por eso que se utilizan mucho como herramientas de *pentesting*[22], pero queda en manos de la persona experta en seguridad en saber configurarlo e interpretar después los resultados.

Como puntos débiles debemos destacar el alto número de falsos positivos que arrojan las herramientas DAST, el extenso tiempo requerido para realizar cada análisis, y la necesidad de realizar el análisis al final del ciclo de desarrollo. Como no encuentran fallos en líneas concretas del código, estas herramientas tampoco son capaces de dar consejos de programación segura al equipo de desarrollo, y por eso muchas veces se complementan los análisis DAST con otro tipo de herramientas y medidas de seguridad. Es común combinar herramientas DAST y SAST, o utilizar una solución IAST que ya incorpore parte de las dos metodologías en una misma herramienta.

Al realizar sus análisis sobre las URL descubiertas por la araña, es posible también que no encuentre todos los recursos que habría que analizar por lo que sería necesario incluir estas direcciones a mano. Existen también muchos riesgos que no se pueden identificar desde el exterior, como ocurre con los ataques a ciegas por inyección SQL o cuando la vulnerabilidad no está identificada públicamente (conocidos como 0-day[52])

22 Metodología para evaluar la seguridad de una infraestructura informática intentando explotar las vulnerabilidades de forma segura

.2.3.2 Selección herramientas DAST

Arachni

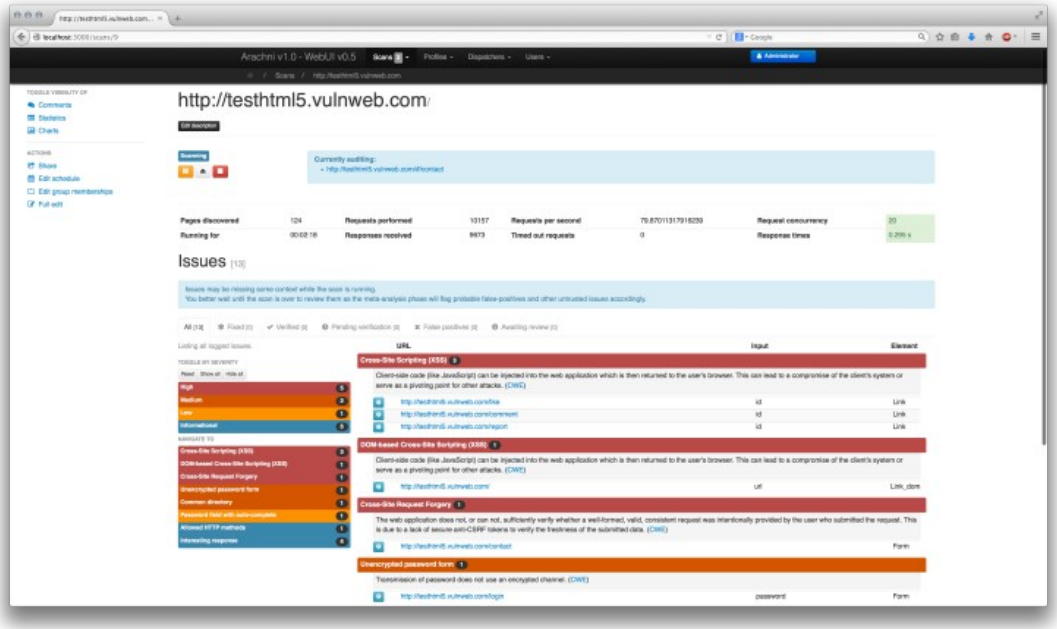


Ilustración 13: Informe web con vulnerabilidades encontradas (fuente: Arachni)

- **Sitio web**: https://www.arachni-scanner.com
- **Plataforma**: GNU/Linux, Windows, MacOSX
- **Fecha última versión**: 29-3-2017
- **Licencia:** Arachni Public Source License[53]
- **Puntos fuertes:**
  - Soporta las tecnologías web más populares (HTML5, JavaScript, AJAX...)
  - Aprende de las respuestas HTTP recibidas y es capaz de evaluar la fiabilidad de los resultados para identificar falsos positivos.
  - Admiten vectores de entrada personalizados ya que puede probar parámetros de entrada inesperados, malformados o aleatorios (*fuzzing*[23]), y es capaz de escanear cualquier parte del protocolo (HTTP u otros).
  - Al estar escrito en el lenguaje de programación Ruby, Arachni es muy escalable y modular.
  - Permite la colaboración entre varias personas usuarias y varias plataformas a la hora de analizar proyectos.
- **Puntos débiles:**
  - El proyecto está sin actualizar ni mantener desde hace cuatro años[54].
  - Aunque permite la integración en otras aplicaciones, la selección de aplicaciones compatibles es menor.
- **Informes:**
  - Los informes se generan en formatos HTML, JSON, YAML y XML.
  - Cuando termina el escaneo, se pueden auditar en la interfaz web los informes.

23 Proceso automatizado para encontrar fallos de software mediante introducción aleatoria de permutaciones de datos

Nikto

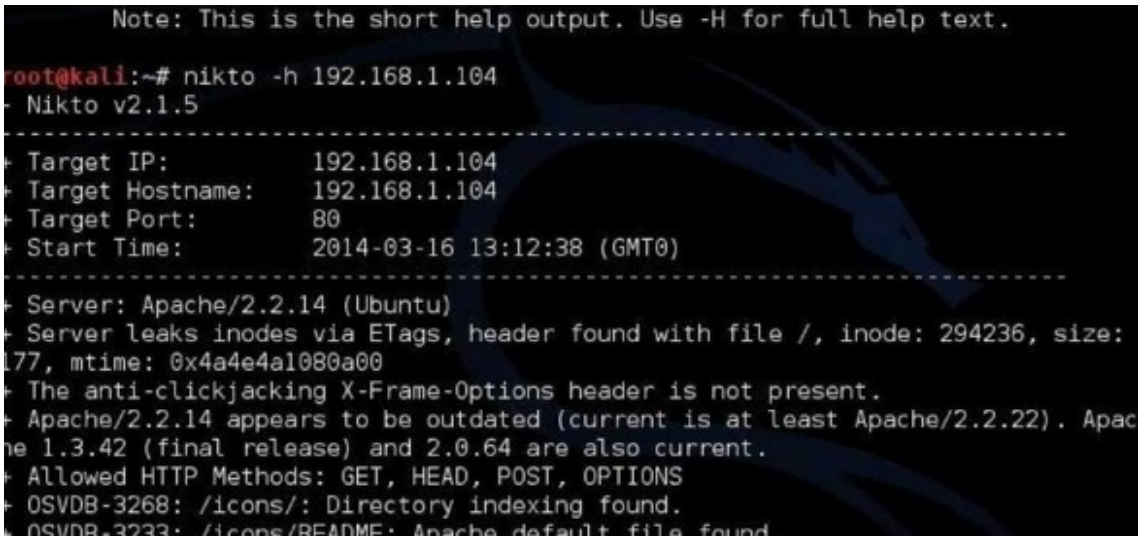


Ilustración 14: Inicio análisis de herramienta Nikto por línea de comandos (Fuente: CyberX)

- **Sitio web**: http://www.cirt.net/nikto2
- **Plataforma**: GNU/Linux
- **Fecha última versión**: 9-7-2015
- **Licencia:** GPL-2.0[55]
- **Puntos fuertes:**
  - Comprueba si existen ficheros o *scripts* malicioso, versiones de servidores desactualizados, así como diversos problemas específicos de los sitios web.
  - Se integra con el escáner de vulnerabilidad comercial *Nessus* para que éste lo invoque cuando encuentre un sitio web.
  - Al estar compuestos de unos pocos *scripts*, su instalación es sencilla.
  - Además de poder personalizar el análisis, *Nikto* puede ser configurado para evadir la detección de algunos IDS.
- **Puntos débiles:**
  - La última versión estable fue lanzada hace 6 años, pero el repositorio en GitHub tiene actualizaciones de las vulnerabilidades para testear más recientes[56].
  - La herramienta está diseñada para ser usada sobre la línea de comandos, aunque existe un *frontend* llamado *Wikto.*
  - No tiene mecanismos para aprender a identificar los falsos positivos, por lo que hay que comprobar los resultados manualmente.
  - Los escaneos suelen incluir mucho “ruido”, pero este comportamiento es necesario para hacer análisis de bajo perfil[57].
- **Informes:**
  - Los resultados con las vulnerabilidades se comparten siguiendo el formato de la base de datos de vulnerabilidades de código abierto (OSVDB[58]).
  - Se puede cambiar el formato del informe de HTML a MSF (formato Metasploit)

Web Application Attack and Audit Framework (W3af)

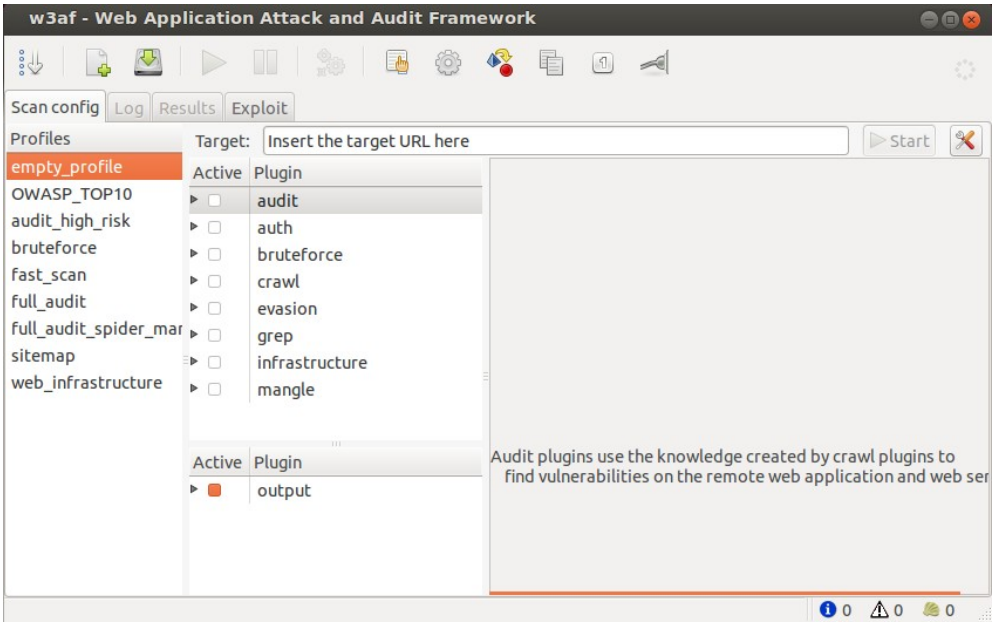


Ilustración 15: Opciones de análisis de w3af (fuente: w3af)

- **Sitio web**: http://w3af.org
- **Plataforma**: GNU/Linux, Windows (excepto la última versión)
- **Fecha última versión**: 7-3-2015
- **Licencia:** GPL v2.0
- **Puntos fuertes:**
  - Detecta más de 200 tipos diferentes de vulnerabilidades (SQLi, XSS, buffer overflow...)
  - Incluye una interfaz gráfica además de herramientas para la línea de comandos.
  - Ha conseguido aglutinar a su alrededor una comunidad amplia de testers de aplicaciones web.
  - Cuenta con capacidades para hacer también análisis de código estático.
  - Permite seleccionar los *plugin* de las vulnerabilidades que queramos probar, reduciendo así el tiempo de análisis.
  - Es una herramienta modular y flexible gracias a la facilidad de integrar *plugins* de Python.
- **Puntos débiles:**
  - Al igual que Nikto, la versión estable del motor lleva tiempo sin ser actualizada, pero los perfiles usados para el análisis se actualizan con más frecuencia.
  - La instalación ha de realizarse a través de la línea de comandos y cuenta con varias dependencias que no siempre será posible resolver en algunos sistemas GNU/Linux.
- **Informes:**
  - Los resultados se ofrecen en forma de árbol que se puede ir desplegando y seleccionando.
  - El ítem seleccionado muestra el tipo de vulnerabilidad, así como la cabecera HTTP con la petición y ubicación del recurso vulnerable.
  - También crea un mapa de sitio con la relación de recursos detectados.

Zed Attack Proxy (ZAP)

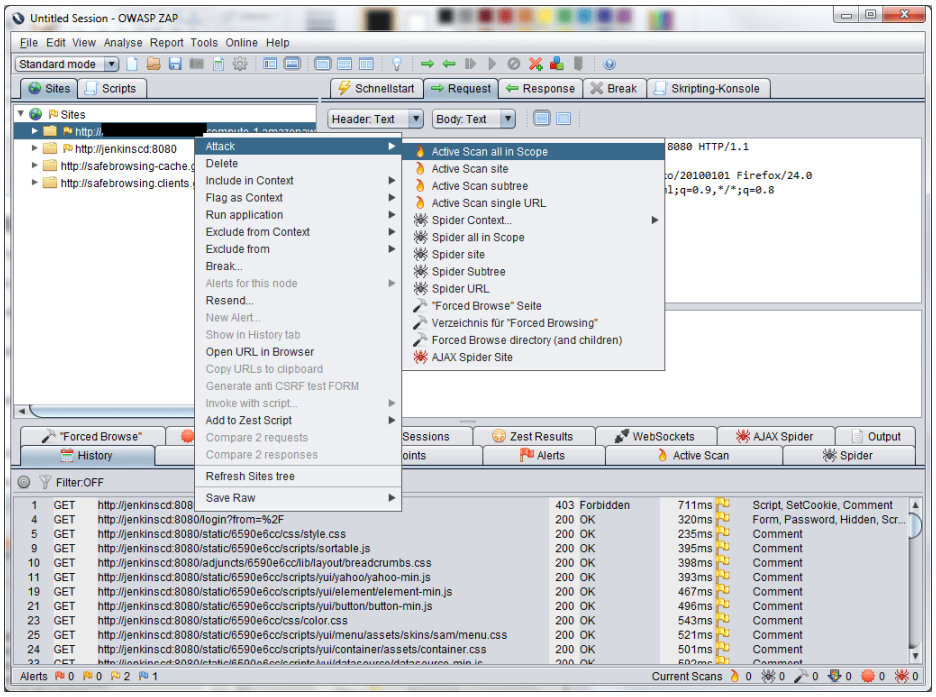


Ilustración 16: Opciones de análisis de ZAP (fuente: ZAP)

- **Sitio web**: https://www.zaproxy.org
- **Plataforma**: GNU/Linux, Windows, MacOSX
- **Fecha última versión**: 30-1-2021
- **Licencia:** Apache-2.0
- **Puntos fuertes:**
  - Es el escáner de web apps más usado, y se actualiza constantemente gracias a las contribuciones de la comunidad.
  - Instalación sencilla, sin apenas dependencias (tan solo requiere Java para funcionar).
  - Interfaz (UI) intuitiva que también incluye opciones avanzadas.
  - Disponibles multitud de tutoriales y vídeos con guías de uso.
  - Análisis automático o manuales para personas expertas que quieran profundizar más.
  - Al igual que Arachni, admite vectores de entrada personalizados
- **Puntos débiles:**
  - Si el proyecto tiene muchos componentes, el análisis es lento. Se recomienda utilizar la opción de escaneo pasivo para no sobrecargar la web.
  - No tiene un buen sistema para gestionar autenticación, control de acceso, manejo de errores o validación de datos de servidores *backend* usados en la aplicación web.
  - Como veíamos en la ilustración nº 12, en el análisis de rendimiento realizado por OWASP la capacidad de detección de su herramienta ZAP era menor que muchas herramientas propietarias, pero el test se hizo hace ya varios años y es posible que el rendimiento actual sea superior.
- **Informes:**
  - El informe muestra todas las posibles vulnerabilidades detectadas, e incluye una referencia con una posible solución.
  - Permite generar informes en distintos formatos.

.2.4 Herramientas para las Pruebas de Seguridad de Apps Interactivas

.2.4.1 Análisis herramientas IAST

Las herramientas IAST son una solución intermedia entre las pruebas de “caja blanca” de SAST y las de “caja negra” de DAST. Como mezcla de ambas, IAST se integra en las fases de desarrollo y pruebas del ciclo de vida de desarrollo de software. La gran diferencia respecto a SAST y DAST es que IAST se inserta en el código y se ejecuta en las pruebas junto con la aplicación analizada, lo que le permite mayor precisión a la hora de localizar y contextualizar los errores de seguridad.

La metodología IAST se implementa de dos formas distintas: de forma activa o pasiva. Ambos requieren un agente insertado en la aplicación, pero la versión activa requiere una herramienta DAST para hacer el escaneo de vulnerabilidades. Su función será reducir los falsos negativos y positivos proporcionados por DAST (que es quien “ataca” a la aplicación). Al depender de DAST, que recordemos se usa al final del ciclo de desarrollo de software una vez esté el sistema completado, la metodología IAST activa no es apropiada para entornos de desarrollo rápidos o ágiles.

En el modo pasivo para las herramientas IAST el agente insertado es independiente y monitoriza y analiza por si mismo el código mientras se ejecuta la aplicación a testear. El agente monitoriza las peticiones y respuestas HTTP, llamadas a la base de datos y otros flujos de datos, y cuando se detecta una vulnerabilidad sabe localizar el punto de código donde ocurre. Las herramientas IAST tienen una tasa de falsos positivos muy baja porque verifican cada vulnerabilidad identificada para asegurarse de que es real y explotable.

Otra de las ventajas es su adecuación para probar APIs, ya que estos suelen tener las pruebas funcionales ya automatizadas.

Como las herramientas IAST se integran en el código de la aplicación a analizar, deben cumplir con los requisitos y lenguaje de programación usados en el proyecto. Eso reduce el número de herramientas IAST disponibles ya que son soluciones muy especificas.

Las herramientas IAST tampoco impiden que ocurra la explotación, por lo que estando activos y a la escucha, si hay un ataque real, lo detectaría y reportaría, pero no lo bloquearía. Por eso a veces se usan también para analizar el tráfico, pero si el objetivo es proteger el sistema es mejor soluciones de auto-protección de la aplicación en tiempo de ejecución (RASP[59]), ya que no solo son capaces de monitorizar el tráfico en tiempo real, sino también bloquear los ataques.

OWASP sólo menciona en su web una herramienta IAST que es gratuita pero no de código abierto, pero sin embargo, la herramienta Enlightn listada en la sección SAST de OWASP puede considerarse un híbrido entre SAST+DAST e IAST, por lo que se procederá a analizar la misma en esta sección.

.2.4.2 Selección herramientas IAST

Enlightn

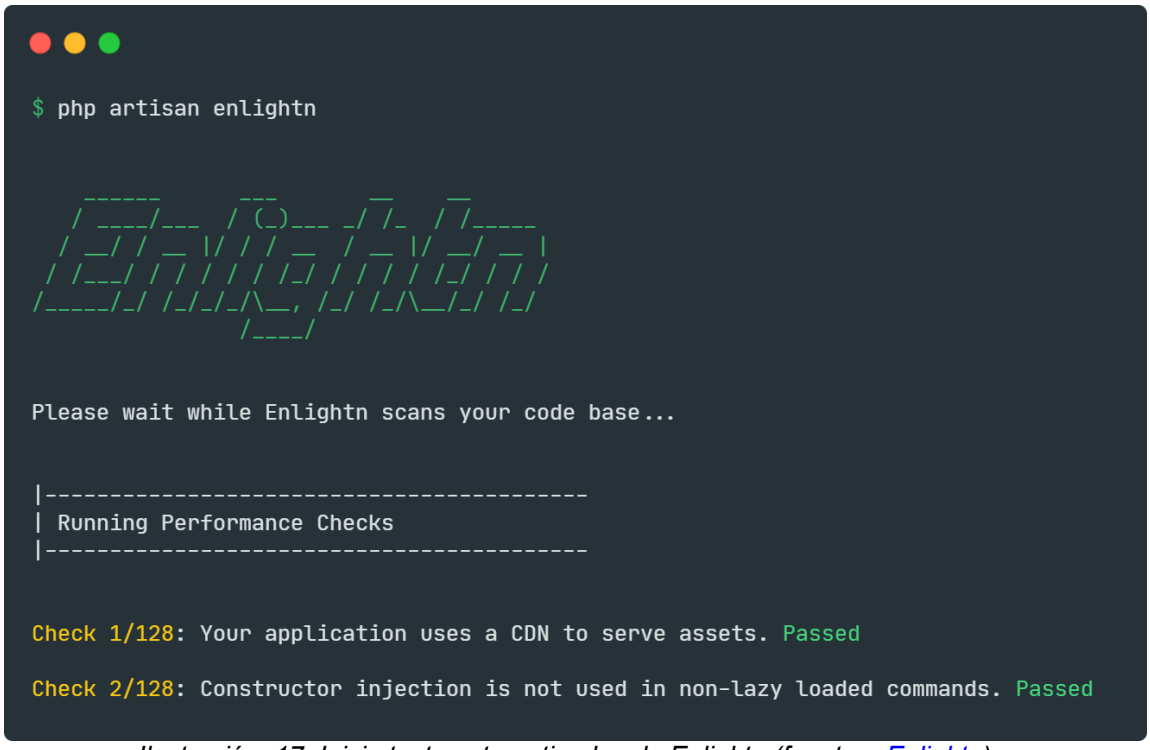


Ilustración 17: Inicio tests automatizados de Enlightn (fuente: Enlightn)

- **Sitio web**: https://www.laravel-enlightn.com
- **Plataforma**: GNU/Linux y MacOSX
- **Fecha última versión**: 23-3-2021
- **Licencia:** LGPL v3[60]
- **Puntos fuertes:**
  - Incluye más de 64 test automatizados para revisar el código o la configuración del servidor.
  - Ofrece recomendaciones en base a los hallazgos para mejorar la seguridad y el rendimiento de la aplicación web, ya que también detecta cuellos de botella.
  - Actualizado con frecuencia.
  - Se integra con el sistema de gestión de versiones Git.
- **Puntos débiles:**
  - Para acceder a todos los tests automatizados (128 en total) es necesario la versión de pago. De los 48 tests de seguridad, 28 son de pago.
  - Es apto para ejecutar antes de desplegar o publicar el proyecto, por lo que para entonces se han podido acumular muchos errores.
- **Informes:**
  - Ofrece una descripción de cada una de las pruebas en las que ha encontrado errores. En la descripción se indica las líneas de código donde ocurre el error, junto con un enlace a documentación para corregirlo.
  - Al terminar todas las pruebas se genera un informe que incluye estadísticas de los hallazgos.

.2.5 Herramientas para el Análisis de la composición del software

.2.5.1 Análisis herramientas SCA

Hoy en día los proyectos de software se construyen utilizando componentes de terceros. En este proyecto nos enfocamos en especial en proyectos que usan componentes de código abierto, como son muchas de las aplicaciones web que se construyen por parte de equipos pequeños de desarrollo. Combinando estos componentes con software propio se consigue lanzar proyectos ambiciosos con menor coste temporal y económico.

Estos componentes que se integran en las soluciones pueden tener vulnerabilidades, y para facilitar su revisión y detección, existen herramientas que analizan estos componentes. Las herramientas SCA realizan análisis automatizados en el código fuente de la aplicación web, detectando en ellos los componentes de código abierto que se estén utilizando y sus versiones. Esas versiones se cotejan para detectar en ellos tanto vulnerabilidades de seguridad como problemas con las licencias, ya que puede haber incompatibilidades entre distintas licencias de software libre.

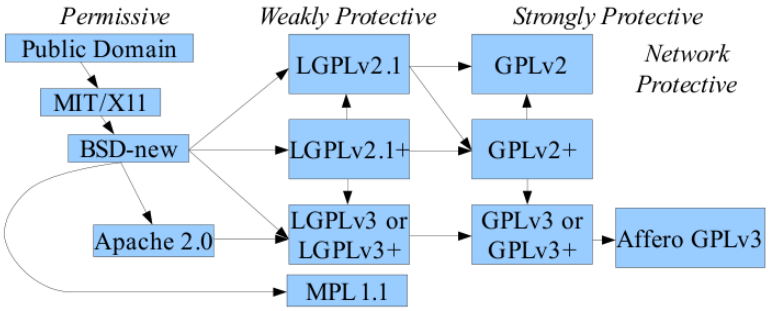


Ilustración 18: La flecha indica que una licencia se puede combinar con la otra, adoptando en conjunto la licencia de la segunda[61] (Fuente: Wikipedia)

Algunas herramientas SCA pueden también ofrecer recomendaciones para corregir las vulnerabilidades detectadas. En algunos casos no hará falta corregir el fallo ya que la herramienta SCA puede indicarnos si se utiliza la función o librería vulnerable en nuestro proyecto web.

Existen soluciones que automatizan todo este proceso, y si se integran en los sistemas de control de versiones, son capaces de alertar de la vulnerabilidad antes de solicitar la descarga de los ficheros. El foco de las herramientas SCA suele estar en la detección de todas las vulnerabilidades, pero cada vez ofrecen mejores resultados priorizándolos en base a la gravedad de la vulnerabilidad (p.ej, indicando su puntuación CVSS[62]), y sugiriendo la corrección (siendo a veces necesario simplemente la actualización del componente de terceros).

Hoy en día las herramientas SCA han madurado y se integran en los repositorio y entornos de desarrollo, facilitando su incorporación al SDLC. La web de OWASP lista varias herramientas entre las que se incluyen algunas desarrolladas por ellos mismos, por lo que en este apartado se analizan varias de ellas.

.2.5.2 Selección herramientas SCA

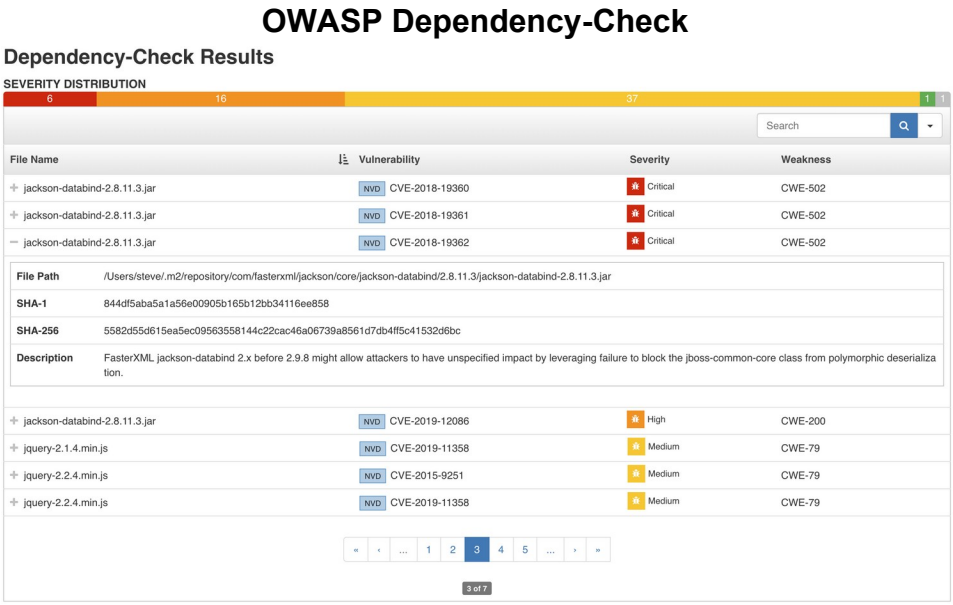


Ilustración 19: Resultados de comprobación de dependencias (fuente: Github)

- **Sitio web**: https://owasp.org/www-project-dependency-check
- **Plataforma**: GNU/Linux, Windows, MacOSX
- **Fecha última versión**: 23-3-2021
- **Licencia:** Apache 2.0
- **Puntos fuertes:**
  - Se integra fácilmente en el proceso de construcción (build) de software.
  - Analiza las dependencias y en base a su enumeración de plataforma común (CPE), devuelve las vulnerabilidades CVE asociadas.
  - La herramienta está siendo constantemente actualizada.
  - El análisis es rápido, y para agilizar el acceso a la la base de datos nacional de vulnerabilidades de los EEUU se puede cachear de forma incremental antes de cada testeo.
  - Aunque en principio está disponible solo para la linea de comandos, existe un *plugin* para realizar y mostrar el análisis de forma gráfica[63].
- **Puntos débiles:**
  - Si la dependencia se ha compilado localmente es posible que el *hash* difiera del oficial, por lo que no identificaría correctamente el componente.
  - Al usar NVD[24] para buscar las vulnerabilidades, existe el riesgo de no detectar vulnerabilidades más recientes que aún no han sido validadas en NVD. Por eso la mayoría de las herramientas SCA son de pago e incluyen bases de datos más actualizadas.
- **Informes:**
  - Los resultados se devuelven en un informe en formato XML o HTML, e incluyen también métricas, tendencias o hallazgos.
  - Las métricas en formato gráfico (con el plugin de Jenkins) son interactivas, pudiendo ver mayor detalle de los hallazgos que engloban.

24 *National Vulnerability Database* es la BBDD de Vulnerabilidades Nacional de Estados Unidos

WhiteSource Renovate

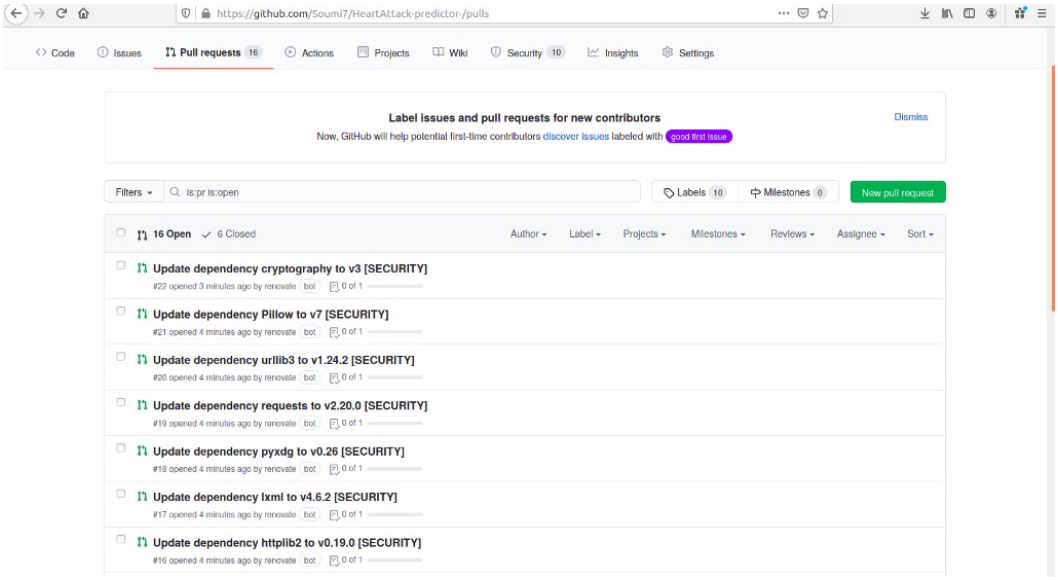


Ilustración 20: Automatización de pull requests después de análisis (fuente: Medium)

- **Sitio web**: https://github.com/renovatebot/renovate
- **Plataforma**: Aplicación web
- **Fecha última versión**: 23-4-2021
- **Licencia:** AGPL 3.0[64]
- **Puntos fuertes:**
  - Permite automatizar las actualizaciones de los componentes dependientes a través de *pull requests*[25] *(PR).* El equipo de desarrollo es responsable de aceptar la solicitud de fusión y probar el software después.
  - Al estar integrado con el un sistema de control de versiones, se pueden ver los cambios que se realizan en el código fuente de los componentes a actualizar.
  - Se integra tanto en la nube (GitHub, Azure DevOps, Bitbucket Cloud, Bitbucket Server, Gitea y GitLab) como de forma local (*on-premise*)
  - El comportamiento del bot que realiza el análisis de forma autónoma es altamente configurable, permitiendo mayor flexibilidad a la hora de integrarlo en el flujo de trabajo
- **Puntos débiles:**
  - Poca documentación, tutoriales y vídeos disponibles para aprender a utilizar la herramienta.
  - Algunas funcionalidades solo están disponibles en la app para GitHub desarrollada por el fabricante, pero no en las versiones instaladas de forma local.
  - Requiere crear una cuenta para su funcionamiento, y aunque de momento ofrecen de forma gratuita un numero ilimitado de repositorios a analizar por el bot, podrían cambiar esas condiciones en un futuro.
- **Informes:**
  - Los resultados se muestran integrados en la plataforma en la que se ha instalado el *bot* de *renovate*, pero se añade un fichero JSON en la raíz del repositorio con el código analizado.

25 En sistemas de control de versiones, es la funcionalidad para comprobar los cambios en una rama antes de fusionarla con otra.

Sonatype DepShield

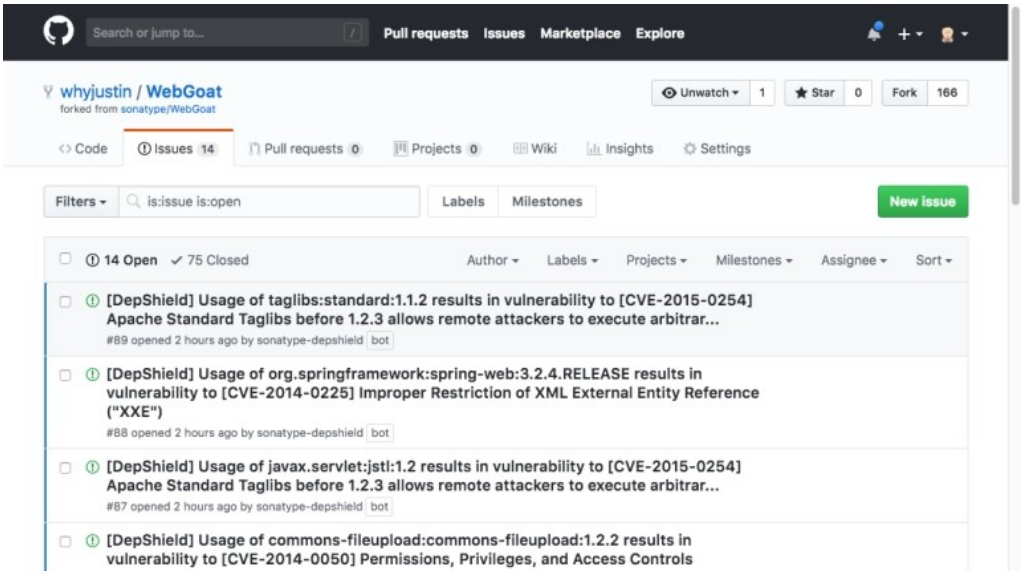


Ilustración 21: Vulnerabilidades identificadas en distintos componentes (fuente: TechCrunch)

- **Sitio web**: https://depshield.github.io/
- **Plataforma**: Aplicación web solo para GitHub
- **Fecha última versión**: 23-4-2021
- **Licencia:** AGPL 3.0
- **Puntos fuertes:**
  - Utiliza el catalogo de componentes de código abierto *Sonatype OSS Index[65]* diseñado para identificar vulnerabilidades y transmitir los niveles de riesgo de los mismos.
  - Supervisa continuamente los proyectos hospedados en GitHub y crea de forma automática propuestas (issues)[26] para detectar vulnerabilidades de seguridad.
  - Permite ver detalles relacionados con la vulnerabilidad, incluyendo CVE y CVSS.
- **Puntos débiles:**
  - Al igual que WhiteSource Renovate, requiere registro y de momento ofrece monitorización continua en repositorios GitHub públicos y privados de forma gratuita, pero pueden cambiar las condiciones en un futuro.
  - La versión gratuita no cuenta con funciones para automatizar las correcciones, tan solo muestra como propuesta lo detectado, y hay que corregirlo de forma manual.
  - Además de *Sonatype OSS Index, l*a versión de pago cuenta con fuentes revisadas y curadas por el fabricante.
- **Informes:**
  - Las vulnerabilidades y recomendaciones se muestran en forma de propuestas de GitHub integrados en la plataforma.

26 Para rastrear ideas, mejoras, tareas o errores del proyecto en GitHub.

OSS Review Toolkit

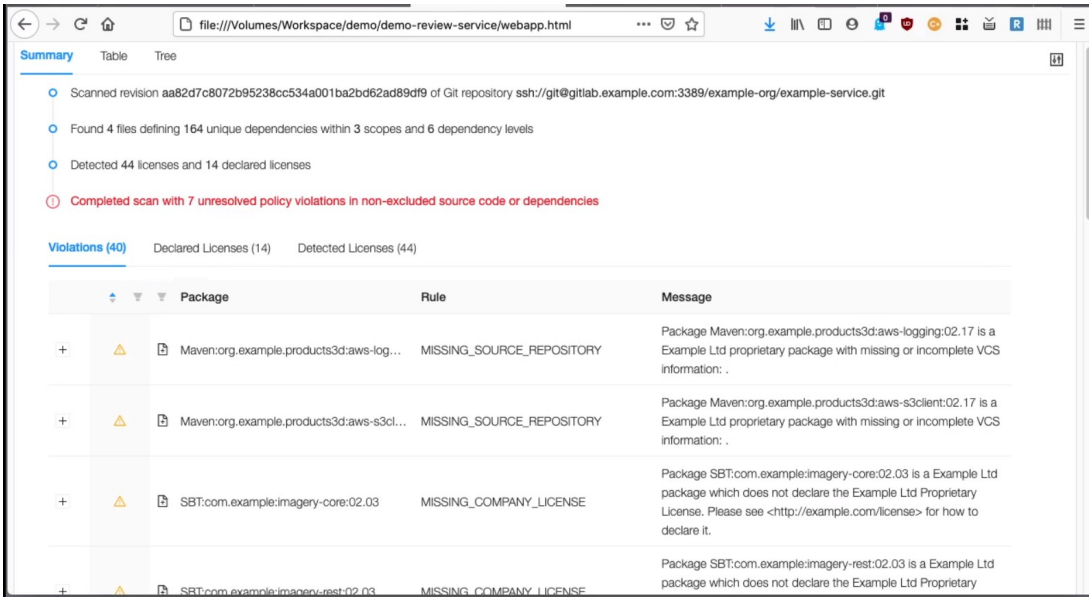


Ilustración 22: Resultados análisis con problemas detectados con varias licencias (fuente: GitHub)

- **Sitio web**: https://github.com/oss-review-toolkit/ort
- **Plataforma**: GNU/Linux, Windows y macOS
- **Fecha última versión**: Código actualizado a abril de 2021
- **Licencia:** Apache 2.0
- **Puntos fuertes:**
  - Especialmente diseñado para analizar y detectar problemas con las licencias de código abierto. Detecta problemas de compatibilidad entre las distintas licencias de los componentes y el proyecto, informando sobre cómo corregir el problema y generando automáticamente el fichero NOTICE[27].
  - La herramienta *advisor* busca en distintos proveedores (configurable) vulnerabilidades conocidas en los componentes del proyectos. También detecta componentes que no estén actualizados apropiadamente
  - Las dependencias de software se detectan a través de los gestores de paquetes de Linux (rpm, apt, yum...), y soporta hasta 11 gestores.
  - Se integra con el IDE JetBrains a través del plugin Kotest[66].
- **Puntos débiles:**
  - La herramienta está más enfocada a analizar problemas con las licencias de software.
  - Documentación poco clara y escasos recursos (tutoriales, vídeos...) para aprender a utilizar la herramienta.
  - Poco optimizado (escrito en lenguaje de programación Kotlin), requiere mínimo de 8GB de RAM y una CPU de 4 núcleos para funcionar.
- **Informes:**
  - El resultado se muestra en formato JSON o YAML, indicando el árbol de dependencias y meta datos sobre los componentes analizados.
  - ORT incluye una herramienta llamada *reporter* para presentar esos datos de forma visual en distintos formatos.

27 Fichero donde se menciona qué partes del código fuente tienen una licencia diferente.

.3. Fase de implementación

.3.1 Integración herramientas en el ciclo de vida de desarrollo de un sistema (SDLC)

.3.1.1 Metodología para reducir falsos positivos

Tanto las pruebas de “caja blanca” como de “caja negra”, así como las comprobaciones en las dependencias de un proyecto de software arrojarán muchos resultados que o bien se referirán a vulnerabilidades que no afectan al proyecto, son erróneas o exageradas, son improbables y difíciles de explotar, o no podemos abordar su corrección.

Ya sea porque el analizador ha cometido un error o por que no nos interesa la advertencia, no es siempre posible revisar todos los resultados, y mucho menos volver a dedicar tiempo a analizarlos en busca de falsos positivos, por lo que tal como veremos en el apartado 3.1.5, la integración de las herramientas SAST, DAST, IAST y SCA se hará utilizando una metodologías de desarrollo de software ágil e iterativo. De esta forma, el número de vulnerabilidades detectado puede mantener un tamaño limitado al no tener que llevar a cabo el análisis una vez que esté todo el proyecto concluido. Recordemos que en la metodología ágil, cada iteración tiene como objetivo obtener un “producto viable mínimo”, es decir, una versión más pequeña y simple del proyecto final pero completamente funcional.

Las herramientas SAST, DAST, IAST y SCA que se utilicen, también han de ser configuradas en la medida que sea posible para acotar el tipo de vulnerabilidades que se busquen, a fin de que se ajusten a la arquitectura y la realidad del proyecto web que queramos securizar. En las herramientas que lo permitan, los resultados que se hayan considerado como falsos positivos en una iteración, también se deberán marcar o añadir a listas blancas para evitar que vuelvan a surgir en futuras iteraciones y análisis. A su vez, esto requiere que en la planificación, en algún momento haya que volver a analizar los falsos positivos a fin de evitar que por error se hayan marcado como tal cuando las herramientas estuvieran detectando vulnerabilidades de suficiente importancia. También se pueden combinar dos herramientas del mismo tipo, integrando una en cada ciclo y dejando la otra para cuando se llegue a un hito (p.ej, antes de publicar una nueva *release* de la aplicación o cada cierto número de iteraciones).

El análisis de los falsos positivos requiere que exista un análisis de riesgo y amenazas previo para que los desarrolladores que tengan que verificarlos, sepan identificar las amenazas que más afecten al proyecto. Siguiendo un proceso iterativo es más fácil entrenar y conseguir que el equipo encargado de analizar los falsos positivos desarrolle suficiente intuición para identificar de forma más rápida en cada ciclo las vulnerabilidades que haya que descartar.

El equipo de desarrollo también debería haber recibido formación sobre prácticas de programación seguras, teniendo en cuenta algunos de los

aspectos que la OWASP recomienda en este ámbito para las aplicaciones web, y cada ciertos ciclos sería conveniente verificar el grado de acierto que se ha tenido tanto en la identificación de falsos positivos, como en la reducción total de vulnerabilidades detectadas debido a la mejora de la calidad del código seguro. Este proceso iterativo combinado ayudará a reducir el *backlog* de vulnerabilidades y facilitará su tratamiento.

En proyectos donde se incorporen componentes antiguos y poco actualizados, las herramientas arrojarán muchos falsos positivos, por lo que la solución que se propondrá en el apartado 3.1.5 tratará de evitar este escenario asegurando que gracias a una herramienta SCA se evite utilizar dependencias que causen vulnerabilidades.

.3.1.2 Integración herramientas en un entorno de desarrollo integrado

Los entornos de desarrollo integrados, abreviados como IDE (Integrated Development Environment), son paquetes de software que proporcionan interfaces para que las personas desarrolladoras de software escriban código, organicen grupos de texto y automaticen las redundancias de la programación. Más allá de un simple editor, los IDE ofrecen múltiples funcionalidades y se integran con las características de los distintos lenguajes de programación que soportan. Normalmente se integran con el compilador, incluyen un *debugger*[28] visual para poder depurar código, añaden sugerencias para completar código, marcan errores de programación en el editor y facilitan la refactorización[29] y organización del código.

Estas herramientas están diseñadas para mejorar la productividad, pero también la mejora de la calidad del código que se escribe ya que permite escribir, mejorar y procesar el código dentro de un único entorno que incluye ciertos elementos asistenciales. Entre estas funcionalidades de asistencia se puede integrar también el resultado de algunas herramientas SAST, IAST o SCA. La incorporación de los resultados de herramientas que analicen código SAST, IAST o SCA directamente en el IDE no interrumpiría ni añadiría complejidad adicional al proceso de desarrollo, ya que se sumaría a las sugerencias de errores que ya proporcionan los entornos de desarrollo. Sí haría falta una política y metodología para atender las vulnerabilidades (limitándose a un porcentaje de los mismos o tratando de planificar y resolver todos), y tal como hemos visto en el apartado 3.1.1, también tendrán que tener en cuenta los falsos positivos que arrojen.

De las herramientas SAST analizadas en el apartado 2.3, tan solo *Horusec* se integra con IDE, pero en este caso proporciona solamente un *plugin* para integrarlo con el entorno de desarrollo de Microsoft Visual Studio que funciona en sistemas Windows. Como veremos más adelante, varias soluciones SAST analizadas se integran con los repositorios de sistemas de control de versiones, permitiendo incorporarlas de otra forma al proceso de desarrollo. La única herramienta IAST analizada, Enlightn, tampoco tiene integración con IDE pero sí con repositorios a través de un bot para GitHub[67]. En el grupo de

28 Aplicación para testear y depurar otras aplicaciones
29 Proceso de reestructuración de código sin cambiar su comportamiento o funcionalidad

herramientas IAST analizadas, solo vemos que *OSS Review Toolkit* se integra con el IDE JetBrains a través del plugin Kotest; al igual que con las herramientas SAST analizadas, el resto tienen mayor integración con los sistemas de control de versiones. IAST tiene que integrar elementos en el código para poder ejecutarse con la aplicación y así analizar y contextualizar los errores detectados; esto limita aún más la oferta de soluciones IAST disponibles ya que son soluciones muy especificas.

Existen multitud de soluciones comerciales que sí se integran con diferentes IDEs, pero en este trabajo tratamos de encontrar soluciones de software libre dirigidas a desarrollos web que se construyen con componentes de código abierto. Esto limita las posibilidades de encontrar soluciones visualmente intuitivas e integradas con otras aplicaciones de entornos de escritorio, primando más las soluciones orientadas a la linea de comandos. En el mundo del software libre es habitual encontrar un mayor enfoque hacia la línea de comandos, y que cada herramienta o proyecto proponga soluciones muy concretas, y que combinando distintas herramientas se llegue a proporcionar las funcionalidades más completas que encontramos en suites comerciales más ambiciosas.

A pesar de no tener soluciones SAST, IAST o SCA integrables a IDEs como Eclipse, Netbeans o JetBrains, la naturaleza de los proyectos de código abierto y los APIs con los que han sido diseñado podría facilitar que aparecieran plugins o extensiones para posibilitar su integración en un futuro.

.3.1.3 Análisis de sistemas de control de versiones

Los sistemas de control de versiones (VCS) son herramientas para la gestión del código fuente de un proyecto de software. Estas herramientas permiten hacer seguimiento y gestionar los cambios que se realizan en el código, guardando cada modificación en una base de datos especial.

Son herramientas especialmente útiles para proyectos en el que intervienen más de una persona, ya que si alguien del equipo introduce un error en el código, el equipo de desarrollo puede comparar versiones anteriores del código para ayudar a solucionarlo, o simplemente recuperar una versión anterior donde el error no estaba presente.

En el escenario que planteábamos para este trabajo de fin de máster contábamos con equipos de unas dos o tres personas que incorporaban distintos componentes de código abierto al proyecto, aumentando la complejidad del mismo. Un sistema de control de versiones nos permitirá trabajar con una versión actual del proyecto sobre el que los distintos miembros del equipo realizarán cambios, con la ventaja de que a modo de “máquina del tiempo” el sistema guardará el estado que tuvo el proyecto en distintos momentos.

Debido a su diseño, estos sistemas ofrecen funcionalidades adicionales que facilitan la gestión del código fuente convirtiéndolas en una herramienta muy potente. Entre las funcionalidades, podríamos destacar las siguientes:

- Realización de copias de seguridad: como los archivos se guardan a medida que se editan, se puede restaurar un fichero o todo el proyecto tal como estaba en la fecha que queramos.
- Habilitar la sincronización del proyecto: distintos miembros del proyecto pueden compartir sus archivos y modificaciones, asegurando que todo el equipo esté al día con la última versión.
- Función “deshacer”: si los cambios realizados causan problemas y no se encuentra la solución, se puede saltar atrás a una versión reciente en la que funcionaba bien.
- Seguimiento de los cambios: junto con la actualización de los archivos se pueden incluir mensajes explicando los cambios realizados. Estos mensajes se guardan en el sistema de control de versiones y no en el código fuente del proyecto. Los mensajes ayudan a contextualizar la evolución que sigue cada fichero.
- Seguimiento de la autoría: cada cambio se registra con el nombre de la persona que lo hizo.
- Creación de versiones de prueba: antes de hacer grandes cambios, se puede trabajar con una versión clonada del proyecto, y cuando funcione bien, se pueden registrar (fusionar) los cambios en la versión principal.
- Ramificación y fusión: además de las versiones de prueba temporales, se pueden bifurcar (*fork*) los proyectos en un área separada y modificarla de forma aislada. Más adelante se podrían fusionar nuevamente, o el *fork* puede continuar de forma autónoma como ocurre con muchos proyectos de software libre.

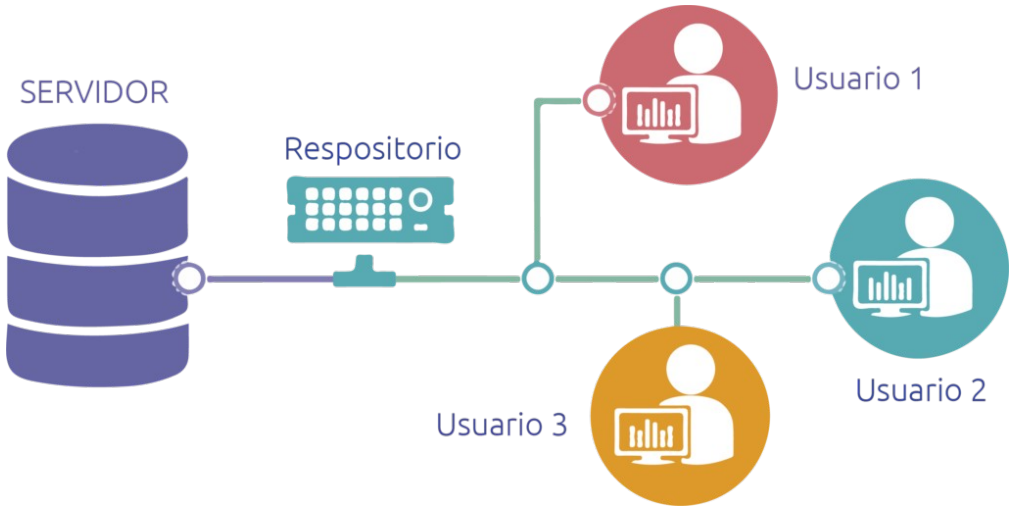


Ilustración 23: Esquema básico de funcionamiento de un VCS (fuente: Nettix)

Aunque existen diferencias entre los distintos productos de software para gestionar código fuente, la mayoría sigue un esquema en el que el código fuente y otros ficheros necesarios del proyecto se guardan en un repositorio que se hospedará en un servidor y se accederá a él a través de un cliente (normalmente, un ordenador personal). El cliente tendrá una copia de trabajo

que se sincroniza desde el tronco o rama principal a un directorio local y desde el que se trabajará modificando los ficheros.

Entre las operaciones básicas que podemos realizar está la posibilidad de añadir un nuevo fichero al sistema para que se haga el seguimiento y control del mismo, verificar el número o código de versión de un fichero, solicitar la versión más reciente, descargar un fichero, subir una versión actualizada de otro archivo (añadiendo o no un mensaje), revisar los cambios que se han realizado sobre un documento o descartar todas las modificaciones hechas y sincronizar nuevamente con la última versión que contenga el repositorio.

Cuando dos o más personas modifican un mismo fichero, se genera un conflicto y aquellas personas que estén autorizados podrán analizar los distintos cambios para aceptar unos u otros a la versión principal, o dejar una de las ramas separada. Distintos sistemas de control de versiones manejan la gestión de conflictos de forma diferente, como veremos a continuación.

De entre los sistemas de control de versiones que se distribuyan con licencia de código abierto, podemos destacar CVS, Subversion, Git y Mercurial.

Concurrent Versions System (CVS)[68]

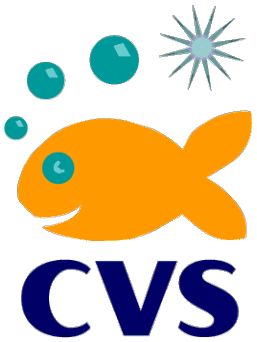


Ilustración 24: Logotipo de Concurrent Versions System (CVS) (fuente: CVS)

Es el más antiguo dentro del mundo del código abierto y por tanto, uno de los más conocidos en entornos veteranos. Al igual que otros sistemas, CVS guarda *snapshots o* copias completas del proyecto en un momento determinado (podría ser análogo a una copia de seguridad).

Debido a su antigüedad y popularidad, CVS funciona en muchas plataformas (GNU/Linux, Windows, Mac OS X...), tiene detrás una comunidad robusta que ha generado mucha documentación y ayuda, y cuenta con un cliente para la línea de comandos completo que facilita la creación de *scripts.*

Aunque sigue evolucionando y añadiendo nuevas funcionalidades, presenta varios problemas en su diseño. Por ejemplo, el coste de realizar operaciones en otras ramas es computacionalmente alto, y no soporta operaciones atómicas por lo que en caso de error antes de completar la transacción (p.ej, desconexión de la red) puede generar corrupción de datos en la base de datos del repositorio.

Apache Subversion (SVN)[69]

Ilustración 25: Logotipo de Apache Subversion (SVN) (Fuente: Subversion)

SVN es considerado como el sucesor de CVS, ya que corregía algunos de sus fallos o carencias a la vez que mantenía la compatibilidad. Mientras que CVS tiene una licencia GNU (una licencia más orientada al movimiento del software libre), SVN tiene licencia Apache (una licencia con una aproximación más cercana a la filosofía del código abierto[70]).

Para evitar que la base de datos que se guarda en el repositorio se corrompa, SVN incluye la funcionalidad de realizar operaciones atómicas: al solicitar el registro de los cambios, se guardan todos o ninguno, evitando así que se guarden cambios parciales tal como ocurría con CVS. También mejora el problema que tenía CVS con la creación de ramas y su mantenimiento, ya que SVN está diseñado para soportar múltiples ramas a modo de árbol, y que estos perduren en el tiempo.

Sin embargo, es más lento que CVS a la hora de comparar distintas versiones de los ficheros, y no permite el control de versiones distribuido donde pueda haber más de un repositorio (uno por cada cliente) que luego se sincronicen con un repositorio central. SVN, al igual que CVS, también tiene problemas cuando se renombran o mueven ficheros y directorios, ya que no se suele guardar como una nueva versión.

Mercurial[71]

Ilustración 25: Logotipo de Mercurial (Fuente: Mercurial)

Mercurial es un sistema de control de versiones distribuido, y está desarrollado en Python en vez de en lenguaje C. En cuanto a funcionalidades y funcionamiento es similar a SVN, lo que hace que junto con su completa documentación facilite la migración de un sistema a otro.

Es un sistema ligero que tiene buen rendimiento y ofrece escalabilidad, e incorpora una interfaz web que facilita su gestión. Sin embargo, no permite fusionar desde dos ramas padre[30] diferentes.

Quizá la estrategia de ser sencillo y similar a SVN es la que hace que Mercurial sea más modesto que otros sistemas distribuidos como Git, que veremos a continuación.

Git[72]

Ilustración 26: Logotipo de Git (Fuente: Git)

Git fue desarrollado por Linus Torvalds (autor inicial del kernel de Linux) cuando la empresa detrás de BitKeeper[73], el sistema de control de versiones propietario que se usaba para gestionar el código fuente de Linux, empezó a poner impedimentos a las condiciones con las que el equipo de desarrollo del kernel de Linux podía usar el mismo de forma gratuita[74].

Desde el inicio se diseño siguiendo una estrategia distinta a CVS o SVN, ya que la experiencia de trabajar en el kernel de Linux (el mayor proyecto de desarrollo de software que existe[75]) primero sin sistema de control de versiones, luego con CVS, y después con BitKeeper, hizo que Linus Torvalds conociera bien las necesidades que debería cubrir este nuevo sistema. Un proyecto como Linux necesitaba un sistema de control de versiones rápido y distribuido, incluyendo herramientas para navegar y gestionar eficientemente el histórico de cambios y versiones que se guarden en los repositorios. Cada instancia hospedada de forma local en los clientes guarda todo el árbol histórico de cambios, algo que permite trabajar contra el repositorio aunque no haya conexión a Internet.

Al igual que SVN, también es ligero y eficiente a la hora de realizar operaciones de ramificación, pero debido a sus potencial, Git es un sistema mucho más complejo de comprender y utilizar. Aún así, Git es uno de los sistemas más populares, sobre todo en proyectos de código abierto[76].

.3.1.4 Integración herramientas en sistemas de control de versiones

De los cuatro sistemas de control de versiones analizados en el apartado anterior, a pesar de tener una curva de aprendizaje mayor, Git es el sistema

30 La rama padre es la rama en la que te encuentras cuando creas la nueva rama.

más completo, más compatible con otras herramientas, y más popular entre la comunidad de desarrolladores de código abierto. Muchos de los proyectos de software libre se encuentran en GitHub[77], plataforma de alojamiento de código para el control de versiones y la colaboración propiedad de Microsoft[78], y eso facilita encontrar en Internet de forma gratuita documentación, tutoriales, ejemplos y foros donde poder resolver dudas. Del grupo de soluciones SAST analizadas, podemos integrar con repositorios Git las herramientas Bandit y Horusec.

Agnitio Security Code Review Tool y SearchDiggity solo funcionan en Windows como aplicación de escritorio y no permiten automatizar el análisis o integrarlos en ningún sistema de control de versiones. Son útiles como herramientas complementarias, ya que de forma externa y sin necesidad de integrarse, permiten realizar la revisión de código. Su dependencia a plataformas Windows y la obligatoriedad de realizar el análisis de código de manera manual lo hacen menos apropiado para proyectos web basados en componentes de código abierto.

Por su parte, Automated Software Security Toolkit (ASST) está pensado para ser instalado en un entorno XAMPP y aunque tampoco se integra con ningún sistema de control de versiones, el hecho de que busque vulnerabilidades web basados en el Top 10 de OWASP lo hace muy atractivo para aplicaciones web que corran también sobre un entorno XAMPP (aunque de momento solo para aquellos proyectos basados en PHP ya que el sporte para Perl o Python aún no se ha añadido).

Bandit se integra con el sistema de control de versiones distribuido Git, y analiza código escrito con el lenguaje Python. Para automatizar el análisis es necesario utilizar Jenkins[79], un servidor de automatización de proyectos de software que facilita los procesos de compilación, testeo y despliegue de los mismos. Jenkins es un proyecto de código abierto y en un sistema GNU/Linux como Ubuntu se puede instalar directamente agregando el repositorio de Debian. Primero añadiendo su clave y luego incluyendo la URL del repositorio al fichero *sources.list*[31]:

```
$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |
sudo apt-key add -

$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

Una vez añadido el repositorio, basta con actualizar el listado de paquetes e instalar el servidor Jenkins y sus dependencias:

```
$ sudo apt update
$ sudo apt install jenkins
```

El servidor se puede lanzar con el comando responsable de examinar y controlar el sistema *systemd*[32] y el administrador de servicios:

31 Fichero que se utiliza para localizar archivos del sistema de distribución de paquetes en sistemas basados en Debian.

32 Una suite de software que proporciona una serie de componentes de sistema para los sistemas operativos GNU/Linux.

```
$ sudo systemctl start jenkins
```

A partir de entonces Jenkins se puede configurar de forma gráfica accediendo desde un navegador a la URL http://IP_nuestro_servidor_Jenkins:8080. Una vez configurado (contraseña de administrador, plugins a instalar...), elegimos la opción *new item,* añadimos el nombre que queramos e introducimos la URL de GitHub donde esté alojado el proyecto. Estas URL suelen tener el formato *https://github.com/Nombre_Cuenta/Nombre_Proyecto/Proyecto.git.* En la pestaña «Source Code Management» habrá que incluir las credenciales con las que se puede acceder al repositorio.

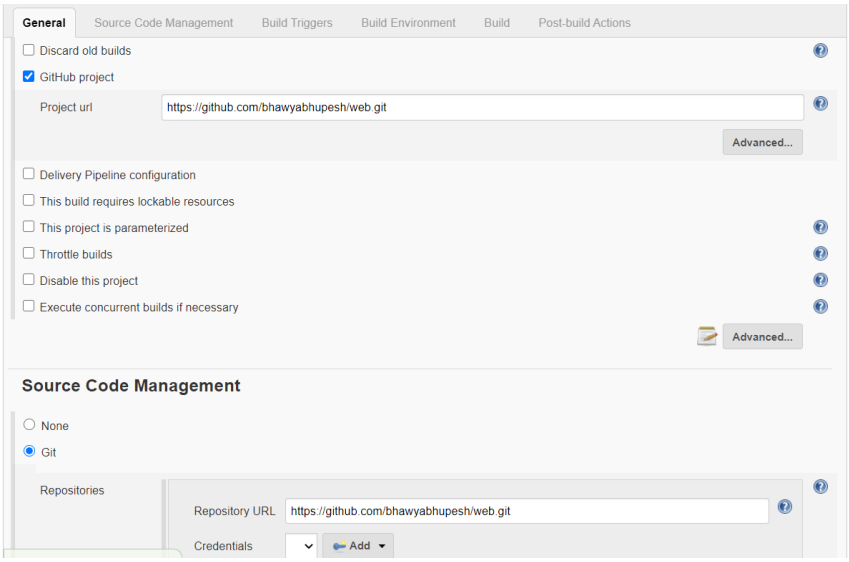


Ilustración 27: Opciones de configuración de Jenkins para bandit (fuente: Medium)

En la pestaña Build le indicamos la forma en que se ejecuta el proceso de compilación; en sistemas GNU/Linux sería desde el *shell* mientras que en sistemas Windows sería desde la herramienta PowerShell. En ambos casos, el comando que se ejecuta en el *shell* sería *«bandit -r $PWD»,* y a partir de entonces se ejecuta la opción *build* de *Jenkins* para que realice el análisis SAST.

Al igual que hemos visto con *bandit, Horusec* también podemos integrarlo con *Jenkins[80]* pero permite además integrarla directamente en GitHub utilizando la función GitHub Actions del servicio de hospedaje que permite automatizar, personalizar y ejecutar los flujos de trabajo de desarrollo de software directamente en el repositorio[81].

En la pestaña *actions* del repositorio de *GitHub* elegimos la opción *new workflow* (nuevo flujo de trabajo) y añadimos el siguiente código en formato YAML:

```
name: SecurityPipeline

on: [push]

jobs:
  horusec-security:
```

```
    name: horusec-security
    runs-on: ubuntu-latest
    steps:
    - name: Check out code
      uses: actions/checkout@v2
    - name: Running Horusec Security
      run: |
        curl -fsSL https://horusec.io/bin/install.sh | bash
        horusec start -p="./" -e="true"
```

A partir de entonces podemos realizar el análisis SAST desde nuestro repositorio en GitHub (p.ej, de forma manual, o configurable para cuando descargamos el código fuente con el comando *pull* o cada vez que realizamos una compilación), aunque de momento limitan la ejecución de las acciones (que se ejecutan en el propio servidor de GitHub) a unas 33 horas al mes para repositorios privados. Si el proyecto hospedado en GitHub es de código abierto, y por tanto, público, no habría limitaciones.

La única herramienta IAST analizada, Enlightn, nos permite realizar en su versión grauita hasta 64 tests automatizados, y podemos integrarlo con Github a través de un Bot, pero en esta versión solo podríamos hacerlo en repositorios públicos. Para poder instalarlo en un repositorio privado haría falta la versión Enlightn Pro, lo que además también daría acceso al doble de test automatizados.

Para poder tener el Bot instalado, primero hay que instalar Enlightn de forma local. Usando el gestor de paquetes *composer* podemos ejecutar el siguiente comando para instalarlo:

```
$ composer require enlightn/enlightn
```

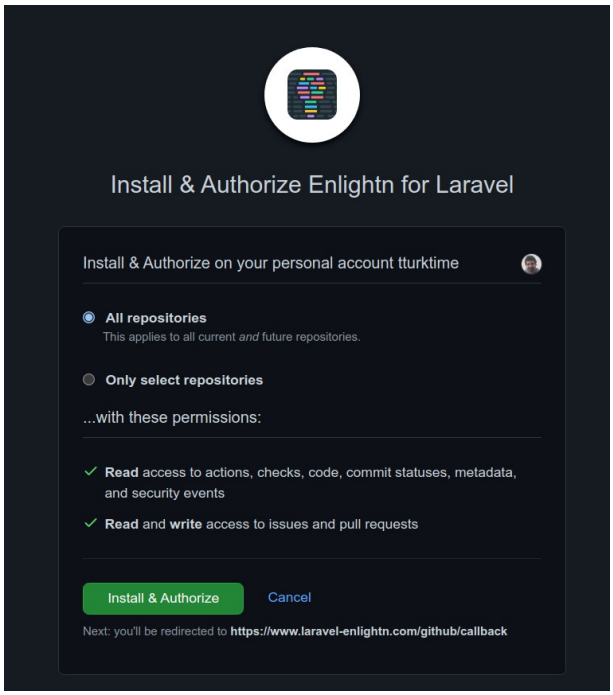


Ilustración 28: Solicitud de permisos antes de instalar (fuente: GitHub)

Con la herramienta ya instalada, debemos ir al apartado de repositorios de la web de Enlightn, que una vez lo autoricemos, posibilitará que podamos integrarla en los repositorios que le indiquemos. Es necesario tener una cuenta (gratuita o de pago) para realizar este primer paso. El segundo paso, tal como hemos visto con *Horusec*, requiere añadir una acción a GitHub con el siguiente código YAML:

```
- name: Run Enlightn Checks and Trigger the Enlightn Bot
  if: ${{ github.event_name == 'pull_request' }}
  env:
    ENLIGHTN_USERNAME: ${{ secrets.ENLIGHTN_USERNAME }}
    ENLIGHTN_API_TOKEN: ${{ secrets.ENLIGHTN_API_TOKEN }}
    ENLIGHTN_GITHUB_REPO: ${{ github.repository }}
  run: |
    cp .env.example .env
    php artisan enlightn --ci --report --review --issue=$
{{ github.event.number }}
```

Para que la acción dentro de GitHub sea funcional, hay que añadir las credenciales de Enlightn y la URL del repositorio en el fichero *config/enlightn.php.* Con la acción configurada, el Bot será capaz de analizar el código y añadir comentarios siempre que se haga una petición de validación (*pull request*) contra el repositorio. En este tipo de peticiones, el cliente desde donde se han realizado modificaciones en el código propone que esos cambios se revisen y acepten por otra persona a fin de que esa contribución termine fusionada en la rama del repositorio desde el que se esté trabajando.

El Bot que analiza el código utiliza como *backend* un servidor de Enlightn para funcionar, pero no se envía código sino metadatos necesarios.

De las cuatro herramientas SCA analizadas, OSS Review Toolkit es la única que no se puede integrar en algún sistema de control de versiones para automatizar el análisis, pero como veíamos en el apartado 3.1.2, era la única herramienta SCA que se se integraba con un entorno de desarrollo integrado.

El proyecto de OWASP Dependency Check tiene al menos dos extensiones que le permiten conectarse con sistemas de control de versiones. Por un lado, hay una extensión para integrarlo con Azure DevOps[82], un producto de Microsoft que además de ofrecer sistema de control de versiones, también ofrece otras funcionalidades. Por otro lado, tenemos una extensión[83] para integrar Dependency Check con GitLab[84], otro gestor de repositorios Git.

Para poder crear una acción en GitLab similar a las que hemos creado en GitHub, se utiliza la imagen Docker que incluye la aplicación y un código YAML que se inserta en GitLab como Jobs[85]. Docker es una plataforma abierta para distribuir aplicaciones. En la web de GitLab diponemos de una imagen de Docker con la aplicación necesaria, una especie de copia completa de la aplicación que al ser iniciada generaría un contenedor[33].

33 Paquete que incluye todos los elementos necesarios para ejecutar la aplicación en cualquier sistema operativo compatible con Docker.

En el código YAML podemos ver cómo se ejecuta el *script dependency-check.sh* que está incluido en la imagen de *Docker* ubicada en *GitLab.* El script escanea el proyecto entero y comprueba la puntuación CVSS de las vulnerabilidades que va encontrando, generando un informe en formato JSON como salida.

```
owasp_dependency_check:
  image:
    name: registry.gitlab.com/gitlab-ci-utils/docker-dependency-
check:latest
    entrypoint: [""]
  stage: test
  script:
    # Job will scan the project root folder and fail if any
vulnerabilities with CVSS > 0 are found
    - /usr/share/dependency-check/bin/dependency-check.sh --scan "./" --
format ALL --project "$CI_PROJECT_NAME" --failOnCVSS 0
    # Dependency Check will only fail the job based on CVSS scores, and
in some cases vulnerabilities do not
    # have CVSS scores (e.g. those from NPM audit), so they don't cause
failure.  To fail for any vulnerabilities
    # grep the resulting report for any "vulnerabilities" sections and
exit if any are found (count > 0).
    - if [ $(grep -c "vulnerabilities" dependency-check-report.json) -gt
0 ]; then exit 2; fi
  allow_failure: true
  artifacts:
    when: always
    paths:
        # Save the HTML and JSON report artifacts
      - "./dependency-check-report.html"
      - "./dependency-check-report.json"
```

Para la herramienta WhiteSource Renovate, el fabricante nos proporciona una app para GitHub que se instala desde la web y solo hay que indicarle con qué repositorios de nuestra cuenta de GitHub queremos integrarlo. Funciona de forma gratuita tanto en repositorios públicos como privados, y se puede configurar para que cuando lo ejecutemos, compruebe las dependencias del proyecto y genere peticiones de validación (*pull requests*) para que quien esté encargado decida cuando y cómo fusionar las actualizaciones.

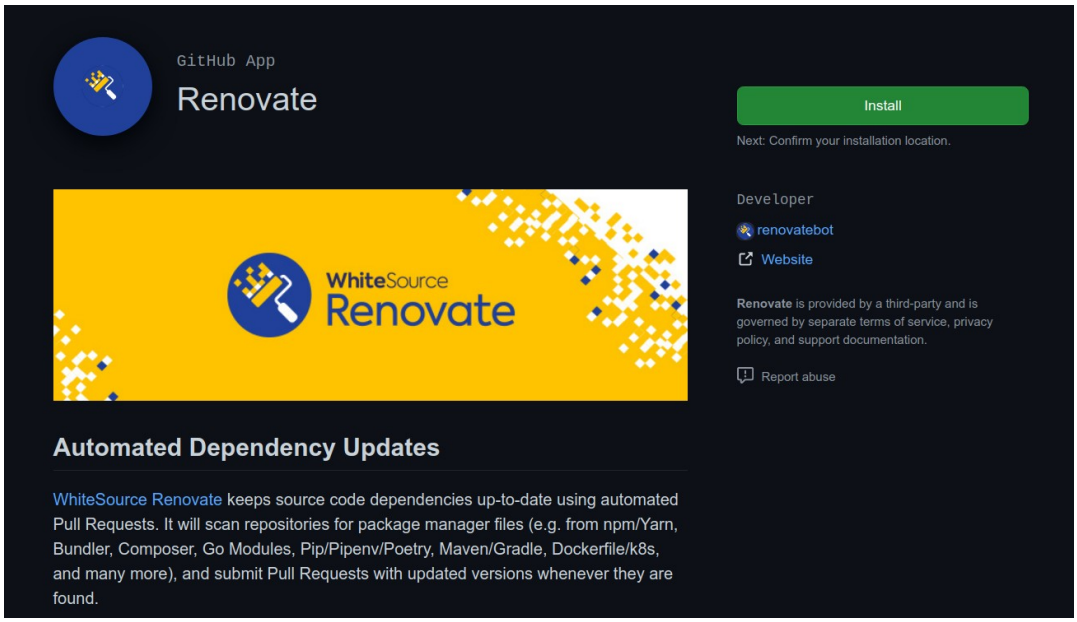


Ilustración 29: Instalación via web de la App de renovate para GitHub (fuente: GitHub)

La última herramienta SCA, Sonatype DepShield, también tiene app para GitHub[86] para integrarla a nuestros repositorios. La app ofrece monitorización continua y genera propuestas donde menciona las vulnerabilidades de seguridad encontradas.

.3.1.5 Propuesta de integración dentro del ciclo de desarrollo y despliegue de una aplicación

En este apartado aunaremos las mejores soluciones y metodologías analizadas durante este capítulo de implementación, seleccionando a su vez algunas herramientas del capítulo de investigación que encajen con dichas metodologías.

Para contextualizar y probar mejor la efectividad de las herramientas SAST, DAST, IAST y/o SCA seleccionadas, el ejemplo de proyecto de software que se va a usar en esta propuesta es de un sitio web real. Esta web está desarrollada en WordPress para una pequeña asociación que realiza proyectos de cooperación y que ofrece información sobre su trabajo en la misma. La web utilizada como ejemplo se adapta al contexto planteado para este trabajo, ya que además de WordPress (y por tanto, programado en PHP usando una base de datos MySQL), tiene varios componentes de código abierto instalados para completar la funcionalidad del sitio, y ha sido desarrollado por un equipo de pequeño de desarrolladores que no había incluido ninguna herramienta para analizar automáticamente el código de la aplicación web.

Si bien la incorporación de herramientas propuestas puede integrarse en un ciclo de desarrollo de software tradicional, en cascada, su implementación es mucho más manejable y efectiva cuando lo hacemos en un modelo de desarrollo ágil. Por eso, el siguiente esquema tan solo representa una iteración, y en cada fase se resume las herramientas de análisis que se utilizan y los pasos que se deben dar a fin de conseguir que tras cada iteración el proyecto web esté más securizado.

La “ensalada” de herramientas seleccionada está basada en soluciones implementadas y adoptadas oficialmente por la propia OWASP. Aunque alguna de ellas aún no tenga una implementación y nivel de madurez suficiente como para competir con otras herramientas analizadas, el apoyo y acompañamiento de la fundación OWASP ayudarán que los proyectos tengan visibilidad y reconocimiento dentro de la comunidad que sustenta y afronta la ciberseguridad, y sigan mejorando y creciendo. Este apoyo para con su mantenimiento es esencial ya que todas estas herramientas dejan de ser útiles una vez se encuentren desactualizadas y dejen de encontrar las vulnerabilidades más recientes.

Ilustración 30: Esquema de todo el ciclo de vida integrando soluciones para securizarla

Como veíamos en en ella análisis de metodologías de desarrollo de software del apartado 1.7, la ventaja de la metodología ágil es que podemos abordar el proyecto de forma iterativa reduciendo el alcance de las funcionalidades del sitio web y por tanto, el número de errores y vulnerabilidades que habría que detectar y corregir en cada iteración. De esta forma, el tratamiento de los falsos positivos se hace más manejable, y se pueden incorporar las recomendaciones planteadas en el apartado 3.1.1 para reducirlos.

La integración de herramientas SAST, DAST, IAST y/o SCA en el SDLC afecta a todas sus fases, sobre todo en una metodología ágil donde estaremos continuamente retroalimentando cada fase con las experiencias vividas en las previas. En el esquema mostrado en la ilustración n.º 30 planteamos una propuesta de incorporación de tres herramientas de análisis de código y vulnerabilidades. Dentro de las herramientas IAST recomendadas por la OWASP tan solo había una listada, y no encajaba en el tipo de proyecto web planteado para este trabajo, por lo que las herramientas que finalmente se proponen son las siguientes:

- OWASP Automated Software Security Toolkit como analizador estático (SAST), herramienta enfocada al escaneo de vulnerabilidades web basados en el Top 10 de OWASP
- OWASP Zed Attack Proxy (ZAP) como analizador de vulnerabilidades (DAST)
- OWASP Dependency-Check como analizador de componentes (SCA)

Lamentablemente estas herramientas no se integran dentro de ningún IDE y deben ser usadas de modo local, haciéndolas menos intuitivas que otras soluciones.

Fase de planificación

Todo proyecto comienza con una buena planificación, y cuando tratamos de incorporar una perspectiva de programación segura al mismo, debemos estudiar las fortalezas y debilidades del equipo que abordará el desarrollo, y tratar de reforzarlas antes de embarcarse en la misma. Una formación, si quiera breve, en buenas prácticas de programación segura como las planteadas en el apartado 2.1 es necesario siempre, y en un proceso iterativo podría hacerse más liviano al poder enfocarse con más detalle en los escenarios y vulnerabilidades más probables para con el proyecto web a desarrollar. Eso hace necesario que después de cada iteración se analicen bien los resultados de cada herramienta de análisis para que al inicio de cada nuevo ciclo se pueda compartir el *know-how* de forma eficiente y rápida, y los efectos de ella tenga reflejo en el resto de fases del cliclo.

Fase de análisis de requisitos

Durante la primera iteración se debe tomar la decisión de qué herramientas incorporar, que en este caso serán ASST, ZAP y Dependency-Check. En todas las iteraciones hay que analizar las librerías o componentes que queramos incorporar, aprovechando lo aprendido en cuando a seguridad en base a los

resultados que arrojaron las herramientas SAST, DAST y SCA, y la forma en que se corrigieron o atendieron las recomendaciones.

Fase de diseño

Una vez seleccionados los componentes que conformarán la solución en cada iteración, sabiendo además qué problemas de seguridad pudieran traer, se deben diseñar mecanismos para minimizar estos riesgos. En la primera iteración se podría trabajar sobre un prototipo básico para hacer el primer análisis de componentes. En el resto de iteraciones ya tendríamos una versión funcional a la que podríamos incorporar nuevos componentes a fin de poder ver si son seguros.

Fase de implementación

Aunque no lo tengamos integrado en IDE, la herramienta SAST se puede ejecutar bajo la línea de comandos y nos muestra los resultados en un fichero HTML. La ejecución de la herramienta se puede realizar durante o al final de la fase, y se recomienda ejecutar OWASP ASST en el servidor de producción en el que esté la web ejecutándose o en uno con una configuración similar, ya que también comprueba si las versiones de PHP y MySQL del servidor están desactualizadas o no. En este trabajo no ha sido posible realizar una prueba SAST en un entorno de producción, por lo que algunos resultados relacionados con la configuración del sistema podrían ser diferentes.

La instalación de ASST es sencilla ya que tan solo requiere un entorno XAMPP para funcionar. En un sistema como Ubuntu, se puede instalar y configurar mysql y php con los siguientes comandos:

```
$ sudo apt install mysql-server

$ sudo mysql_secure_installation

$ sudo apt install php
```

ASST también necesita el entorno de tiempo de ejecución de JavaScript Node.js, que se instala y configura de la siguiente manera (la versión 12.13.0 es compatible con ASST pero podría ser también más reciente):

```
$ sudo apt-get install nodejs -y

$ sudo apt-get install npm -y

$ sudo npm install n -g

$ sudo n 12.13.0
```

A partir de ahora, basta con poner nuestro proyecto web en la carpeta *htdocs* (en Ubuntu está en */var/www/html)* junto con la propia herramienta ASST que podemos descargar/clonar con *git:*

```
$ sudo git clone https://github.com/OWASP/ASST
```

Dentro de la carpeta ASST tenemos que configurar el fichero *config.js* indicando en la variable *DEFAULT_PROJECT_PATH_TO_SCAN* la ruta (relativa o absoluta) donde se encuentra nuestro proyecto Web, y en el fichero *config_php_lang.js* habrá que especifirar algunos parámetros referentes a nuestro servidor MySQL (IP, contraseña de root…). Una vez configurado, lanzamos el script para realizar el análisis:

```
$ sudo node /var/www/html/ASST/main.js
```

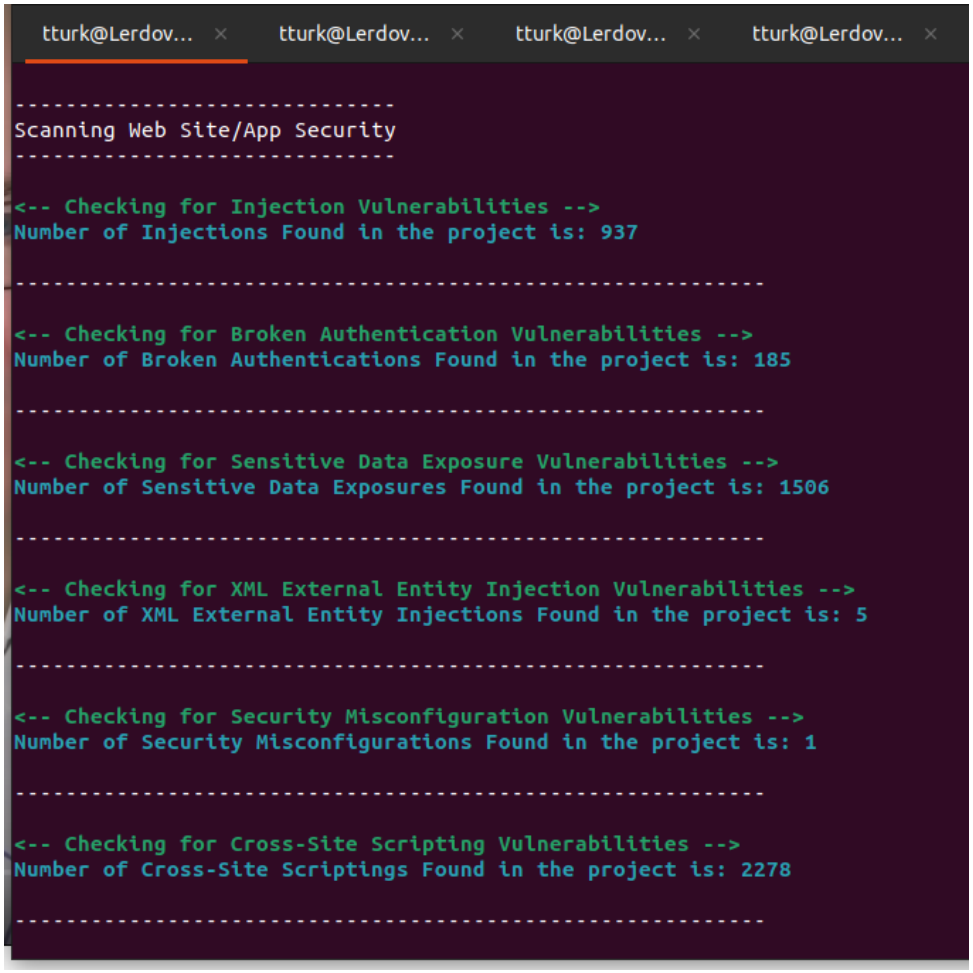


Ilustración 31: Resultados parciales mostrados durante el análisis SAST

A partir de ahora ASST escanea el proyecto web y comprueba todos y cada uno de los archivos línea a línea en busca de vulnerabilidades de seguridad. Las vulnerabilidades detectadas aparecen indicadas en el informe, y clickando podemos saber más acerca del problema encontrado, con recomendaciones de cómo corregir el problema y securizar el sistema.

Queda fuera del alcance de este proyecto analizar las vulnerabilidades encontradas en el proyecto web usado de ejemplo, así como tampoco compete en este trabajo proponer soluciones especificas a estas vulnerabilidades, pero se comentan algunos hallazgos a modo de ejemplo del tipo de problemas que aparecen en sitios web basados en WordPress como el analizado en esta propuesta.

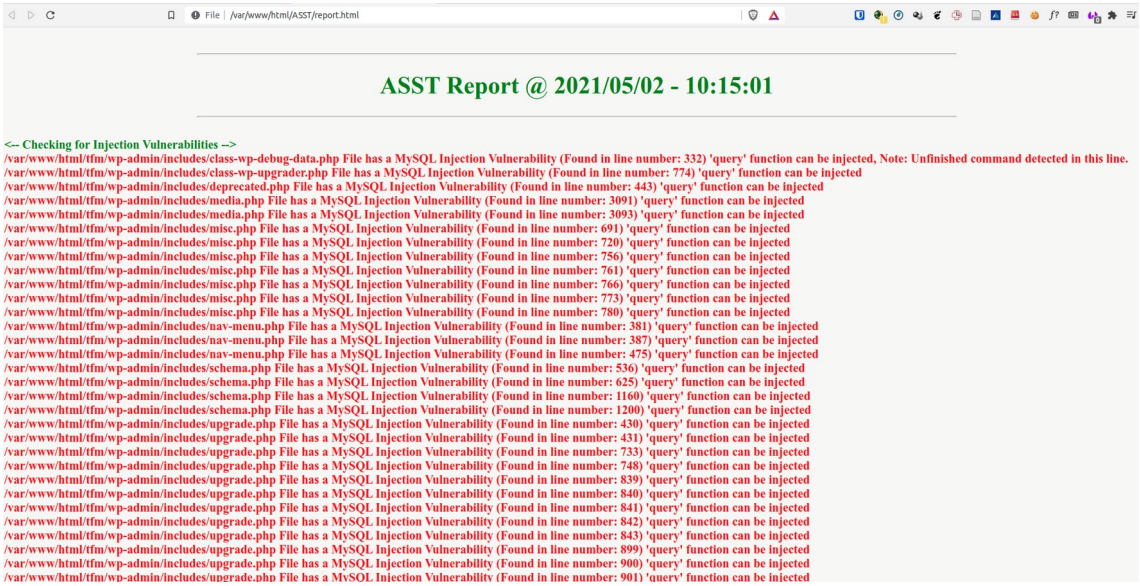


Ilustración 32: Informe completo generado por ASST

Como se puede apreciar en parte en la ilustración de arriba, la mayoría de las vulnerabilidades de inyección detectadas están en los ficheros PHP que vienen con WordPress, y el resto se encuentran en los *plugins* que también se instalaron a la hora de construir la web. El proyecto web analizado fue desarrollado hace más de un año y la versión de WordPress sobre la que está construida la web es la 5.3.x, de la que se conocen algunas vulnerabilidades. La última versión de WordPress, la 5.7.2, corrige muchos de los problemas encontrados pero para poder actualizar habría que comprobar que todos los componentes y *plugins* con los que ha sido diseñada la web, así como la plantilla que se ha usado, siguen siendo compatibles con la última versión de WordPress. En total, esta página web contaba con 937 posibles vulnerabilidades de inyección.

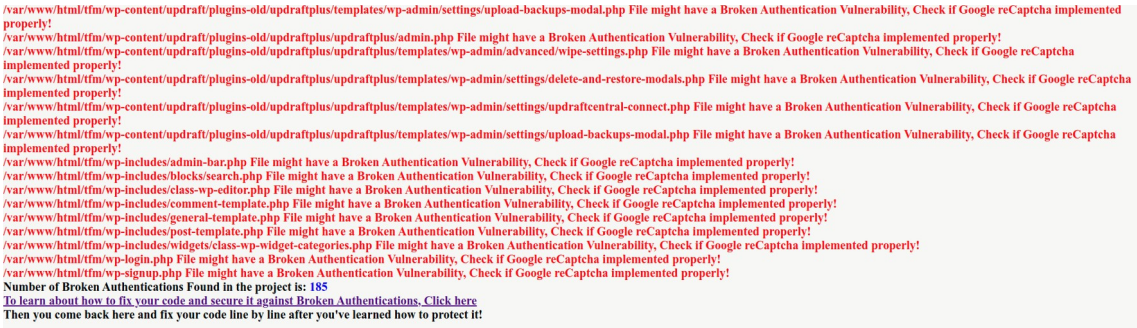


Ilustración 33: Detalle con número de vulnerabilidades e información adicional de ASST

En cuanto a vulnerabilidades que atacantes puedan aprovechar para hacerse pasar por usuarios o usuarias legítimas, ASST fue capaz de encontrar 185. Este tipo de vulnerabilidades se encontraron en plugins de WordPress conocidos como *akismet* (para moderar spam), *contact-form* (para realizar formularios de contacto), *polylang* (para ofrecer una web bilingüe) o *updraft* (para realizar copias de seguridad de la web).

El motor de ASST fue capaz de encontrar también 1506 vulnerabilidades de la exposición de datos sensibles (tanto en WordPress como en los *plugins*

instalados), 5 Vulnerabilidades de inyección de entidades externas XML (todos ellos en varios plugins instalados en Wordpress) o hasta 2278 vulnerabilidades de secuencias de comandos en sitios cruzados (XSS).

En este caso el número de vulnerabilidades es enorme, pero esto se debe a que el análisis SAST se realizó sobre un proyecto web ya completo. En cada iteración habríamos obtenido un número menor y más manejable de avisos.

En esta fase de implementación también se debe realizar un análisis SCA. Al ser el proyecto web privado, se ha optado por realizar el análisis de componentes también a nivel local, sin contar con la integración de la herramienta de OWASP dependency-check en sistemas de control de versiones. Además, GitHub no permite repositorios privados en su versión gratuita, y GitLab tiene otras limitaciones en su servicio gratuito, por lo que no ha sido posible comprobar la integración. La herramientas de OWASP dependency-check también se ejecuta desde la línea de comandos, pero no requiere configuración; basta con descargar y descomprimir la última versión desde *https://github.com/jeremylong/DependencyCheck/releases/download/v6.2.0/dependency-check-6.2.0-release.zip* y ejecutar en análisis sobre la carpeta donde tengamos nuestro proyecto web asignando al análisis el nombre que queramos:

```
$./dependency-check.sh --project "TFM" --scan "/var/www/html/tfm/"
```

```
[INFO] Finished Node.js Package Analyzer (0 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Created CPE Index (1 seconds)
[INFO] Finished CPE Analyzer (1 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (0 seconds)
[INFO] Finished Node Audit Analyzer (5 seconds)
00:00  INFO: Vulnerability found: bootstrap below 3.4.1
00:00  INFO: Vulnerability found: bootstrap below 3.4.0
00:00  INFO: Vulnerability found: bootstrap below 3.4.0
00:00  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.1
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.1
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:04  INFO: Vulnerability found: bootstrap below 3.4.0
00:07  INFO: Vulnerability found: jquery below 3.0.0-beta1
00:07  INFO: Vulnerability found: jquery below 2.2.0
00:07  INFO: Vulnerability found: jquery below 3.4.0
00:07  INFO: Vulnerability found: jquery below 3.5.0
00:07  INFO: Vulnerability found: jquery below 3.5.0
00:07  INFO: Vulnerability found: jquery-ui-dialog below 1.12.0
[INFO] Finished RetireJS Analyzer (9 seconds)
[INFO] Finished Sonatype OSS Index Analyzer (0 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Analysis Complete (18 seconds)
[INFO] Writing report to: /home/tturk/Downloads/dependency-check/bin/./dependency-check-report.html
```

Ilustración 34: Resultados parciales durante análisis SCA

En la misma carpeta en la que hemos ejecutado el script se genera un informe en formato HTML con todos los hallazgos encontrados:

How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: TFM

Scan Information (show all):

- *dependency-check version*: 6.2.0
- *Report Generated On*: Mon, 3 May 2021 12:26:16 +0200
- *Dependencies Scanned*: 1101 (859 unique)
- *Vulnerable Dependencies*: 16
- *Vulnerabilities Found*: 36
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bootstrap.min.js		pkg:javascript/bootstrap@3.3.7	MEDIUM	4		3
bootstrap.min.js		pkg:javascript/bootstrap@3.0.3	MEDIUM	4		3
browserslist:4.12.0		pkg:npm/browserslist@4.12.0	moderate	1		3
browserslist:4.6.3		pkg:npm/browserslist@4.6.3	moderate	1		3
dialog.min.js		pkg:javascript/jquery-ui-dialog@1.11.4	MEDIUM	1		3
elliptic:6.5.2		pkg:npm/elliptic@6.5.2	MEDIUM	4		3
hosted-git-info:2.7.1		pkg:npm/hosted-git-info@2.7.1	moderate	1		3
hosted-git-info:2.8.8		pkg:npm/hosted-git-info@2.8.8	moderate	1		3
ini:1.3.5		pkg:npm/ini@1.3.5	HIGH	2		3

Ilustración 35: Informe completo producido por dependency-check

Como resultado del análisis, se han detectado en este proyecto 16 dependencias vulnerables con 36 vulnerabilidades en ellas, algunas de ellas de severidad grave. Una de las vulnerabilidades grave se encuentra en la conocida herramienta de JavaScript Lodash; en el informe se nos indica que las versiones anteriores a 4.17.19 son vulnerables a ataques de contaminación de prototipo[87]:

Published Vulnerabilities

NPM-1523 suppress

Versions of `lodash` prior to 4.17.19 are vulnerable to Prototype Pollution. The function `zipObjectDeep` allows a malicious user to modify the prototype of `Object` if the property identifiers are user-supplied. Being affected by this issue requires zipping objects based on user-provided property arrays.

This vulnerability causes the addition or modification of an existing property that will exist on all objects and may lead to Denial of Service or Code Execution under specific circumstances.

Unscored:

- Severity: low

References:

- Advisory 1523: Prototype Pollution - - [HackerOne Report](https://hackerone.com/reports/712065) - [GitHub Issue](https://github.com/lodash/lodash/issues/4744)

Vulnerable Software & Versions (NPM):

- cpe:2.3:a:*:lodash:\<4.17.19:*:*:*:*:*:*:*

Ilustración 36: Detalle con información acerca de vulnerabilidad en Lodash

Los resultados obtenidos en ambos análisis deben ser tratados apropiadamente, añadiendo aquellos errores que deban ser corregidos a un sistema de seguimiento de bugs, y aprovechando los escenarios vulnerables encontrados para preparar los casos de prueba para la siguiente fase:

Fase de pruebas

En esta fase, con el proyecto web en una versión MPV, es cuando lanzamos nuestro análisis DAST. La instalación de OWASP ZAP es también trivial. Nos descargamos el script instalador desde *https://www.zaproxy.org/download/* y lo instalamos con permisos de administrador:

```
$ chmod +x ZAP_2_10_0_unix.sh

$ sudo ./ZAP_2_10_0_unix.sh
```

En la primera ventana que nos aparece al abrir el *script zap.sh* instalado, si elegimos que la sesión de ZAP sea persistente, la información analizada será guardada en una base de datos. Podemos elegir el nombre o dejar que use el *timestamp* actual:

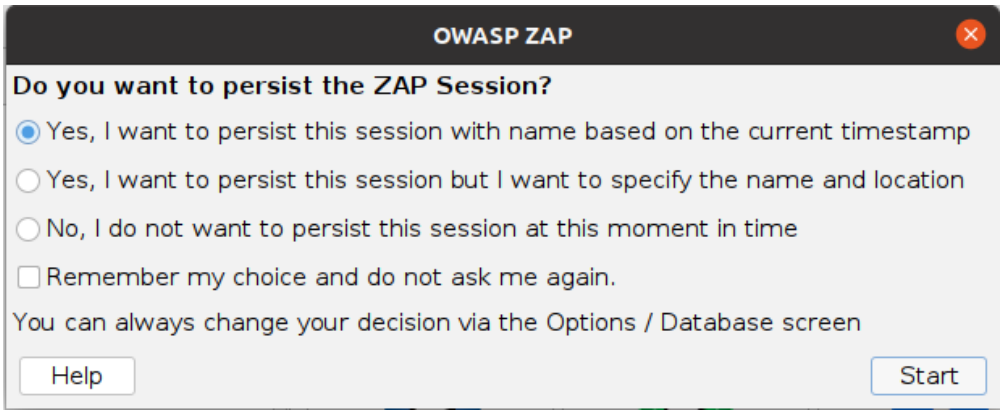


Ilustración 37: Configuración de la sesión de OWASP ZAP

El funcionamiento de ZAP es muy intuitivo, ya que cuenta con interfaz gráfica y es un proyecto muy utilizado y actualizado. Es importante utilizar este tipo de herramientas en aquellas web para las que tenemos permiso para realizar el análisis de «caja negra», ya que ZAP realiza ataques reales para comprobar las vulnerabilidades y podría causar daños. Para evitar posibles daños, se puede ejecutar ZAP en modo seguro, pero en este caso lo ejecutaremos en modo standard a fin de poder localizar más vulnerabilidades en el sitio web.

Desde la pestaña de “Quick scan” podemos elegir la opción de escaneo automatizado (Automated Scan), y una vez le añadimos la URL que nos interesa analizar, debemos elegir con qué navegador se quiere realizar el rastreo y pruebas. Una vez seleccionado, basta con elegir la opción “Attack” para iniciar el análisis, que requerirá de cierto tiempo para poder realizar todas las pruebas sin estresar mucho al servidor.

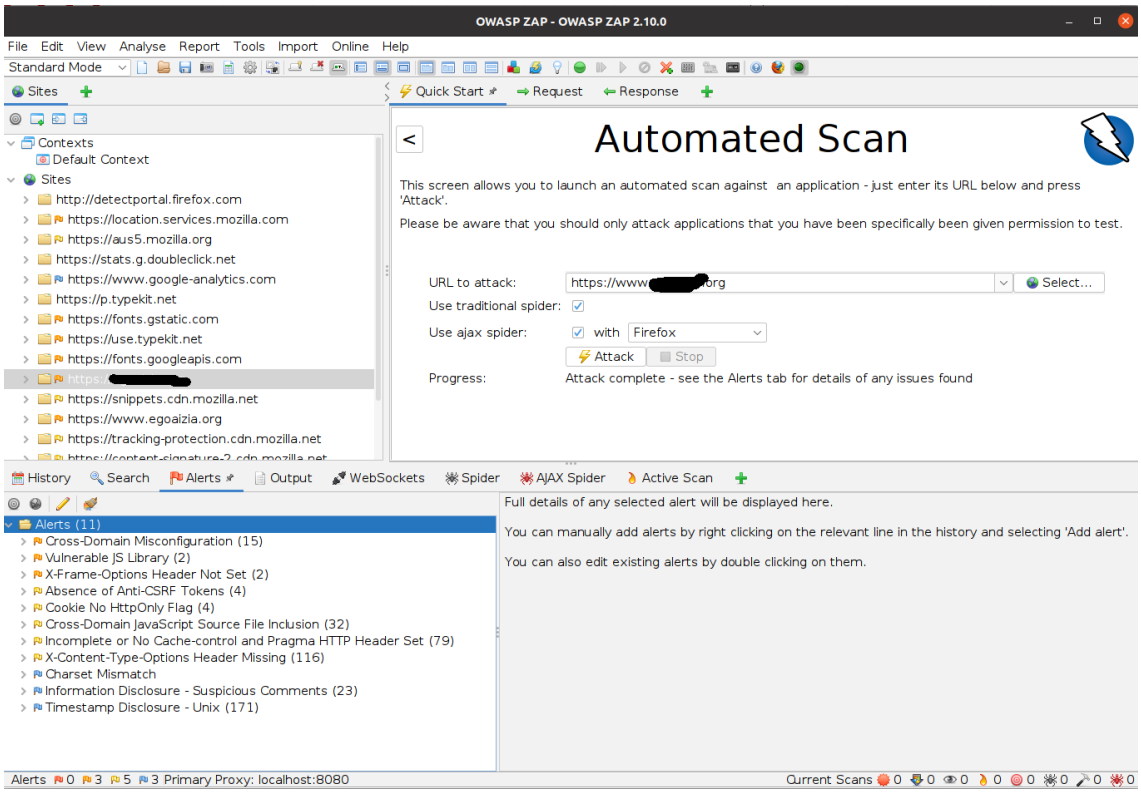


Ilustración 38: Resultado del análisis DAST de OWASP ZAP

El resultado nos muestra 11 alertas con varios tipos de vulnerabilidades encontradas, y por cada tipo encontramos distintas URL afectadas. Estas vulnerabilidades son más difíciles de localizar dentro de nuestro código ya que no se indica la línea donde ocurre (DAST es un tipo de análisis de “caja negra” no es capaz de acceder al código fuente). Sin embargo, el análisis fue capaz de encontrar dos librerías vulnerables usadas en la web que pueden ser fácilmente actualizadas. Estos hallazgos, a diferencia de los más localizados en análisis SAST y SCA, requieren contextualización e intuición, por lo que en cada iteración el equipo de desarrollo debería ser capaz de encontrar aquellos que requieran corrección, e ignorar falsos positivos o hallazgos menos graves.

Fase de despliegue y mantenimiento

La incorporación de las tres herramientas de análisis elegidas no incorpora grandes cambios a estas fases. Sigue siendo necesario añadir mecanismos de seguridad adicionales como todo proyecto web (Firewall, IDS…) y monitorizar la actividad para detectar problemas adicionales. Al estar en un proceso de desarrollo ágil, se puede aprovechar el aprendizaje de cada iteración para reforzar y mejorar la securización de también el entorno usado para desarrollo. Así mismo, es buena oportunidad para revisar toda la iteración y tratar de aprovechar los aciertos y aprender de los errores en lo que respecta a la programación segura de una web.

A quienes estén usando ya el proyecto web (normalmente al final de todas o casi todas las iteraciones) se les puede proporcionar mecanismos para comunicar errores, pero esto dependerá del tipo de servicio contratado con el equipo de desarrollo.

.4. Conclusiones y recomendaciones

Este último capítulo recoge las conclusiones que se han obtenido luego de haber realizado todo el análisis e implementación de este trabajo de fin de máster. También se incluye el grado de objetivos que se han cumplido, así como algunas propuestas que se pudieran realizar a futuro partiendo del trabajo realizado hasta ahora.

.4.1 Conclusiones

El estudio de las distintas herramientas SAST, DAST, IAST y SCA recomendadas por la OWASP, junto con la implementación de una solución basada en proyectos de código abierto para proyectos de la misma naturaleza, ha sido interesante y enriquecedor.

Como se evidenció en el capítulo del análisis de las herramientas, es muy notable la diferencia en cuanto a funcionalidades e integración que ofrecen herramientas comerciales populares frente a las herramientas libres revisadas, pero también es esperanzador saber que se cuenta con una serie de soluciones avanzadas que pueden no solo ser integradas dentro del ciclo de vida de desarrollo de un sistema, sino que gracias a la comunidad que se genera alrededor de este tipo de proyectos, se pueda contar con un producto actualizado y lo suficientemente flexible como para ir adaptándose a las nuevas realidades que van surgiendo en el mundo del desarrollo web.

Este trabajo demuestra la importancia de tener herramientas actualizadas con una fuerte comunidad detrás que las testee, mantenga y mejore. Es por eso que, como veíamos en la propuesta del apartado 3.1.5, se han elegido herramientas de la propia OWASP, ya que durante el análisis del capítulo dos se pudo ver cómo algunas herramientas con mucho potencial dejaron de ser efectivas al no tener mecanismos actualizados para detectar y enfrentarse a vulnerabilidades recientes. Estos tres proyectos seleccionados tienen mucho potencial y gracias a la visibilidad y apoyo de la Fundación OWASP, seguirán creciendo y mejorando tanto en funcionalidades como en integración con otras herramientas. Esta misma filosofía es la que se ha tratado de impregnar en la propuesta que se hace con este trabajos: es necesario mantener una política de actualizaciones de componentes robusta, algo que gracias al software libre y las herramientas analizadas, se puede integrar con cierta facilidad en los procesos de desarrollo.

Las soluciones de código abierto tienen mucha aplicación empresarial, y promueven la mejora de habilidades y competencias de los distintos equipos de desarrollo al requerir una implicación y un conocimiento interno de las herramientas mayor que las que podemos encontrar en soluciones comerciales intuitivas. En este trabajo no ha sido posible medir el grado de disrupción que podría acarrear la implementación de la solución propuesta, pero la realidad es que incorporar las herramientas en el flujo de trabajo tiene como coste tan solo el tiempo y dedicación que se quiera ceder al análisis y corrección de las

vulnerabilidades detectadas, ampliando mucho la horquilla de la que disponemos para minimizar el impacto. Eso sí, a mayor dedicación e integración con todas las fases del SDLC, mayores garantías de que el producto final estará mejor securizado.

Sin lugar a dudas, este trabajo ha sido un ejercicio tanto exigente como provechoso, y ha reforzado mi convicción de que proyectos ágiles, abiertos y colaborativos son la base que hace crecer Internet y el mundo de las aplicaciones y herramientas de software al ofrecer un espacio en el que todas las personas, organizaciones e iniciativas se puedan sumar independientemente de sus ingresos, presupuestos y expectativas.

.4.2 Evaluación de objetivos alcanzados

Con la finalización de este proyecto se han logrado los siguientes objetivos planteados:

- Estudiar un número significativo de herramientas SAST, DAST, IAST y SCA recomendadas desde la OWASP

- Plantear una solución basada en varias herramientas que se puedan integrar de forma ágil al proceso de desarrollo de una web, manteniendo coherencia y equilibrio entre ellas (todas corren sobre GNU/Linux, son proyectos de código abierto, tienen el aval de la OWASP, están orientadas a desarrollos web basados en componentes de código abierto...)

- Facilitar el mantenimiento de componentes de código abierto actualizados gracias a los hallazgos de los análisis.

- Compartir una metodología para aprender buenas prácticas de programación segura, pudiendo realizar el aprendizaje de forma iterativa y apoyado con el *feedback* de las herramientas propuestas.

- Conocer el potencial de las herramientas analizadas para organizaciones pequeñas que apuestan por el software libre o se apoyan en proyectos de código abierto.

Se han encontrado ciertos problemas durante el análisis y la implementación del proyecto, y esto ha podido reducir algunos de los objetivos propuestos. En el proceso de elaboración de este trabajo se evidenció que no había herramientas que cumplieran con todas las expectativas generadas durante el planteamiento de este proyecto. Este hecho muestra una de las debilidades más comunes que podemos encontrar en proyectos de software libre: podemos encontrarnos ante herramientas complejas que requieren más formación y dedicación que otras herramientas comerciales disponibles, y la necesidad de completar algunas funcionalidades con soluciones propias o combinando varias soluciones de código abierto (lo que requiere también de talento y tiempo que era parte de lo que se buscaba evitar al añadir herramientas que asistieran al equipo de desarrollo). A pesar de ello, la solución final es suficientemente

robusta como para ser incorporada tal como está en el desarrollo de muchos proyectos web.

.4.3 Recomendaciones y trabajo futuro

Sería interesante implementar la solución propuesta en un entorno de desarrollo real, y medir si en cada iteración en un proceso de desarrollo ágil se reduce el número y porcentaje de vulnerabilidades detectadas. Un trabajo futuro podría enfocarse en medir el grado de conocimientos del equipo de desarrollo con respecto a la programación segura antes, durante y después de desarrollar un proyecto web usando la metodología propuesta en este trabajo.

Sería necesario también integrar la solución propuesta en este trabajo con los sistemas de control de versiones para comprobar que este tipo de herramientas facilitan la detección y corrección de ciertas vulnerabilidades sin causar gran disrupción al ritmo y carga de trabajo.

En cuanto a las carencias propias encontradas en algunas herramientas, parece evidente que hace falta desarrollar un instalador y configurador sencillo para ASST, e incluso una interfaz gráfica para facilitar que personas que vengan del mundo de Windows y/o las interfaces gráficas, no se encuentren con una barrera incómoda a la hora de utilizar la herramienta.

Personalmente, este trabajo me abre la oportunidad de implementar en el sitio web utilizado en la propuesta algunas de las correcciones sugeridas durante los análisis realizados, para así no solo ayudar a securizar esa web, sino seguir aprendiendo prácticas de programación segura.

5. Bibliografía y fuentes consultadas

En este apartado se listan los artículos, páginas web e informes que se han consultado durante la elaboración de este trabajo. Los enlaces a las páginas web citadas a lo largo de este documento estaban activas y con la información referenciada hasta la última comprobación realizada durante el mes de mayo de 2021.

[1] *Informe «Software Assurance» del Departamento de Seguridad Nacional de los Estados Unidos en referencia a estudio del Instituto de Ingeniería del Software* https://us-cert.cisa.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf

[2] *Ruambo, F. (2019). Network Security: A Brief Overview of Evolving Strategies and Challenges. International Journal of Science and Research (IJSR), 8, 834-841. https://doi.org/10.21275/ART20194980*

[3] Bird, J (2015) 2015 State of Application Security: Closing the Gap, SANS Institute: Reading Room - Analyst Papers.

[4] Relative Cost of Fixing Defects: Dawson, M., Burrell, D., Rahim, E., & Brewster, S. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC). Journal of Information Systems Technology and Planning, 3, 49-53.

[5] *Defect Prevention: Reducing Costs and Enhancing Quality. (2010, febrero 26). ISixSigma. https://www.isixsigma.com/tools-templates/software/defect-prevention-reducing-costs-and-enhancing-quality/*

[6] *FLOSS and FOSS - GNU Project—Free Software Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://www.gnu.org/philosophy/floss-and-foss.en.html*

[7] *Cisco 2015 Annual Security Report. (2015). 53. https://www.cisco.com/c/dam/assets/global/DE/unified_channels/partner_with_cisco/newsletter/2015/edition2/download/cisco-annual-security-report-2015-e.pdf*

[8] *Usage Statistics and Market Share of WordPress, May 2021. (s. f.). Recuperado 22 de mayo de 2021, de https://w3techs.com/technologies/details/cm-wordpress*

[9] *Source Code Analysis Tools | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-community/Source_Code_Analysis_Tools*

[10] *Vulnerability Scanning Tools | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-community/Vulnerability_Scanning_Tools*

[11] *OWASP ZAP Zed Attack Proxy. The ZAP Homepage. (s. f.). Recuperado 22 de mayo de 2021, de https://www.zaproxy.org/*

[12] *Synopsys (2021) Open Source Security And Risk Analysis Report (OSSRA) https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html*

[13] *OWASP Dependency-Check Project | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-dependency-check*

[14] *OWASP Top Ten Web Application Security Risks | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-top-ten*

[15] *Build Security In | CISA. (s. f.). Recuperado 22 de mayo de 2021, de https://us-cert.cisa.gov/bsi*

[16] *SWAMP. (2015, agosto 24). Department of Homeland Security. https://www.dhs.gov/science-and-technology/swamp*

[17] *OWASP in SDLC. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-integration-standards/writeups/owasp_in_sdlc*

[18] Cavusoglu, H., Cavusoglu, H. & Zhang, J. (2006) Economics of Security Patch Management, The Fifth Workshop on the Economics of Information Security (WEIS 2006)

[19] *What is a Minimum Viable Product (MVP)? - Definition from Techopedia. (s. f.). Techopedia.Com. Recuperado 22 de mayo de 2021, de http://www.techopedia.com/definition/27809/minimum-viable-product-mvp*

[20] *Independent Signatories of The Manifesto for Agile Software Development. (s. f.). Recuperado 22 de mayo de 2021, de https://agilemanifesto.org/display/index.html*

[21] *Manifiesto por el Desarrollo Ágil de Software. (s. f.). Recuperado 22 de mayo de 2021, de https://agilemanifesto.org/iso/es/manifesto.html*

[22] *The 2020 Scrum Guide | Scrum Definition. (s. f.). Recuperado 22 de mayo de 2021, de https://scrumguides.org/scrum-guide.html*

[23] *What are sprints? Atlassian. (s. f.). Sprints. Atlassian. Recuperado 22 de mayo de 2021, de https://www.atlassian.com/agile/scrum/sprints*

[24] *Atlassian. (s. f.). Jira | Issue & Project Tracking Software. Atlassian. Recuperado 22 de mayo de 2021, de https://www.atlassian.com/software/jira*

[25] Agile Project Management. (s. f.). Recuperado 22 de mayo de 2021, de https://www.pivotaltracker.com/

[26] Atlassian. (s. f.). Standups for agile teams. Atlassian. Recuperado 22 de mayo de 2021, de https://www.atlassian.com/agile/scrum/standups

[27] *Eric S. Raymond (1999) Release Early, Release Often*. *The Cathedral and the Bazaar,* (s. f.). Recuperado 22 de mayo de 2021, de http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html

[28] *OWASP (2010) Secure Coding Practices Quick Reference Guide https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf*

[29] *A1:2017-Injection | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html*

[30] *A3:2017-Sensitive Data Exposure | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure.html*

[31] *A5:2017-Broken Access Control | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html*

[32] *Adding Salt to Hashing: A Better Way to Store Passwords. (s. f.). Auth0 - Blog. Recuperado 22 de mayo de 2021, de https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/*

[33] *A10:2017-Insufficient Logging & Monitoring | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring.html*

[34] *General Data Protection Regulation (GDPR) – Official Legal Text. (s. f.). General Data Protection Regulation (GDPR). Recuperado 22 de mayo de 2021, de https://gdpr-info.eu/*

[35] *Cross Site Request Forgery (CSRF) | OWASP Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-community/attacks/csrf*

[36] *Source Code Analysis Tools | OWASP. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-community/Source_Code_Analysis_Tools*

[37] *Inc, E. F. (s. f.). The Community for Open Innovation and Collaboration | The Eclipse Foundation. Recuperado 22 de mayo de 2021, de https://www.eclipse.org/*

[38] *Welcome to Apache NetBeans. (s. f.). Recuperado 22 de mayo de 2021, de https://netbeans.apache.org/*

[39] *Apache License, Version 2.0. (s. f.). Recuperado 22 de mayo de 2021, de https://www.apache.org/licenses/LICENSE-2.0*

[40] *Beaton, W. (s. f.). Eclipse Public License 2.0 | The Eclipse Foundation. Recuperado 22 de mayo de 2021, de https://www.eclipse.org/legal/epl-2.0/*

[41] *The GNU General Public License v3.0—GNU Project—Free Software Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://www.gnu.org/licenses/gpl-3.0.en.html*

[42] *T. pip. (s. f.). pip: The PyPA recommended tool for installing Python packages. (21.1.2) [Python]. Recuperado 22 de mayo de 2021, de https://pip.pypa.io/*

[43] *Configuration—Bandit documentation. (s. f.). Recuperado 22 de mayo de 2021, de https://bandit.readthedocs.io/en/latest/config.html*

[44] *PyCQA/bandit. (2021). [Python]. Python Code Quality Authority. https://github.com/PyCQA/bandit (Original work published 2018)*

[45] *Google Hacking Diggity Project Resources—Bishop Fox. (s. f.). Bishop Fox Resources. Recuperado 22 de mayo de 2021, de https://resources.bishopfox.com/resources/tools/google-hacking-diggity/*

[46] *ZupIT/horusec. (2021). [Go]. ZUP IT INNOVATION. https://github.com/ZupIT/horusec (Original work published 2020)*

[47] *MIT License. (2021). En Wikipedia. https://en.wikipedia.org/wiki/MIT_License*

[48] *OWASP/ASST. (2021). [JavaScript]. OWASP. https://github.com/OWASP/ASST (Original work published 2020)*

[49] *Araña web. (2021). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Ara%C3%B1a_web*

[50] *OWASP Benchmark. (s. f.). Recuperado 22 de mayo de 2021, de https://owasp.org/www-project-benchmark/*

[51] *Chen, S. (2017, noviembre 10). Security Tools Benchmarking: WAVSEP 2017/2018 - Evaluating DAST against PT/SDL Challenges. Security Tools Benchmarking. https://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html*

[52] *Zero-day (computing). (2021). En Wikipedia. https://en.wikipedia.org/wiki/Zero-day_(computing)*

[53] *Arachni License. (2014, agosto 17). Arachni - Web Application Security Scanner Framework. https://www.arachni-scanner.com/license/*

[54] *Arachni is no longer maintained. (2020, enero 28). Arachni - Web Application Security Scanner Framework. https://www.arachni-scanner.com/blog/arachni-is-no-longer-maintained/*

[55] *GNU General Public License v2.0—GNU Project—Free Software Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html*

[56] *sullo. (2021). Sullo/nikto [Perl]. https://github.com/sullo/nikto (Original work published 2012)*

[57] *Nikto—An overview | ScienceDirect Topics. (s. f.). Recuperado 22 de mayo de 2021, de https://www.sciencedirect.com/topics/computer-science/nikto*

[58] *Open Sourced Vulnerability Database (OSVDB), (s. f.). Recuperado 22 de mayo de 2021, de http://www.osvdb.org/*

[59] *Jr, J. P. M. (s. f.). What is Runtime Application Self-Protection (RASP)? TechBeacon. Recuperado 22 de mayo de 2021, de https://techbeacon.com/security/what-runtime-application-self-protection-rasp*

[60] *GNU Lesser General Public License v3.0—GNU Project—Free Software Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://www.gnu.org/licenses/lgpl-3.0.en.html*

[61] *The Free-Libre / Open Source Software (FLOSS) License Slide. (s. f.). Recuperado 22 de mayo de 2021, de https://dwheeler.com/essays/floss-license-slide.html*

[62] *NVD - Vulnerability Metrics. (s. f.). Recuperado 22 de mayo de 2021, de https://nvd.nist.gov/vuln-metrics/cvss*

[63] *Jenkinsci/dependency-check-plugin. (2021). [Java]. Jenkins. https://github.com/jenkinsci/dependency-check-plugin (Original work published 2013)*

[64] *GNU Affero General Public License—GNU Project—Free Software Foundation. (s. f.). Recuperado 22 de mayo de 2021, de https://www.gnu.org/licenses/agpl-3.0.en.html*

[65] *Inc, S. (s. f.). Sonatype OSS Index. Sonatype OSS Index. Recuperado 22 de mayo de 2021, de https://ossindex.sonatype.org/*

[66] *Kotest—Plugins | JetBrains. (s. f.). JetBrains Marketplace. Recuperado 22 de mayo de 2021, de https://plugins.jetbrains.com/plugin/14080-kotest*

[67] *Github Bot Integration. (s. f.). Enlightn. Recuperado 22 de mayo de 2021, de https://www.laravel-enlightn.com/docs/getting-started/github-bot.html*

[68] *CVS - Open Source Version Control. (s. f.). Recuperado 22 de mayo de 2021, de https://www.nongnu.org/cvs/*

[69] *Apache Subversion. (s. f.). Recuperado 22 de mayo de 2021, de https://subversion.apache.org/*

[70] *Understanding Open Source and Free Software Licenses. (2018, noviembre 15). Kwork Innovations. https://www.kwork.me/open-source-free-software-licenses/*

[71] *Mercurial SCM. (s. f.). Recuperado 22 de mayo de 2021, de https://www.mercurial-scm.org/*

[72] *Git. (s. f.). Recuperado 22 de mayo de 2021, de https://git-scm.com/*

[73] *BitKeeper. (s. f.). Recuperado 22 de mayo de 2021, de https://www.bitkeeper.org/*

[74] *A Git Origin Story | Linux Journal. (s. f.). Recuperado 22 de mayo de 2021, de https://www.linuxjournal.com/content/git-origin-story*

[75] *Bhartiya, S. (2016, mayo 12). Linux is the largest software development project on the planet: Greg Kroah-Hartman. CIO. https://www.cio.com/article/3069529/linux-is-the-largest-software-development-project-on-the-planet-greg-kroah-hartman.html*

[76] *VI. Source Control. Stack Overflow Developer Survey 2015. (s. f.). Stack Overflow. Recuperado 22 de mayo de 2021, de https://insights.stackoverflow.com/survey/2015*

[77] Build software better, together. (s. f.). GitHub. Recuperado 22 de mayo de 2021, de https://github.com

[78] Microsoft acquires GitHub. (s. f.). Stories. Recuperado 22 de mayo de 2021, de https://news.microsoft.com/announcement/microsoft-acquires-github/

[79] Jenkins. (s. f.). Jenkins. Recuperado 22 de mayo de 2021, de https:/www.jenkins.io/

[80] Installation. (s. f.). Horusec. Recuperado 22 de mayo de 2021, de https://horusec.io/docs/cli/installation/

[81] Features • GitHub Actions. (s. f.). GitHub. Recuperado 22 de mayo de 2021, de https://github.com/features/actions

[82] Azure DevOps Services | Microsoft Azure. (s. f.). Recuperado 22 de mayo de 2021, de https://azure.microsoft.com/en-us/services/devops/

[83] GitLab CI Utils / Docker Dependency Check. (s. f.). GitLab. Recuperado 22 de mayo de 2021, de https://gitlab.com/gitlab-ci-utils/docker-dependency-check

[84] Iterate faster, innovate together. (s. f.). GitLab. Recuperado 22 de mayo de 2021, de https://about.gitlab.com/

[85] Jobs | GitLab. (s. f.). Recuperado 22 de mayo de 2021, de https://docs.gitlab.com/ee/ci/jobs/

[86] Sonatype DepShield (s. f.). GitHub. Recuperado 22 de mayo de 2021, de https://github.com/apps/sonatype-depshield

[87] Guillem, D. S. (2020, septiembre 10). Contaminación de prototipo: Una amenaza infravalorada. MuySeguridad. Seguridad informática. https://www.muyseguridad.net/2020/09/10/contaminacion-de-prototipo/