HackingLab UEx: Aplicación Web Interactiva para la Formación en Seguridad Web

Alberto López Trigo
Universidad de Extremadura
Escuela Politécnica de Cáceres
albertolt@unex.es

Javier Alonso Díaz
Universidad de Extremadura
Escuela Politécnica de Cáceres
javieralonso@unex.es

Agustín Javier Di Bartolo
Universidad de Extremadura
Escuela Politécnica de Cáceres
adibarto@alumnos.unex.es

José Carlos Sancho Núñez
Universidad de Extremadura
Centro Universitario de Mérida
jcsanchon@unex.es

María Mar Ávila Vegas
Universidad de Extremadura
Escuela Politécnica de Cáceres
mmavila@unex.es

***Resumen*—La cantidad de ataques cibernéticos que se producen en la actualidad y sus consecuencias son la razón de que las organizaciones destinen gran parte de sus recursos a la ciberseguridad. La falta de talento en ciberseguridad es un caldo de cultivo perfecto para la proliferación de vulnerabilidades. Este artículo propone el diseño de una aplicación web enfocada a la formación de los desarrolladores, donde puedan aprender sobre programación segura y enfrentarse a los riesgos más comunes de una aplicación web. La aplicación pone a disposición del usuario dos escenarios: uno vulnerable que permite explotar código, y otro seguro ante ataques. El usuario puede comprobar y examinar el código de los dos escenarios, además de poder elegir el lenguaje de programación que se muestra. De esta forma, tiene la oportunidad de comparar el código seguro y el vulnerable en diferentes lenguajes de programación, adquiriendo una doble visión: como *pentester* y desarrollador.**

***Index Terms*—Ciberseguridad, Web App; CTF; OWASP; Seguridad Web, Formación, Software Seguro.**

Tipo de contribución: *Formación e innovación educativa*

I. Introducción

Aunque la tecnología y las aplicaciones web no son nada nuevo y llevan décadas entre nosotros, aún siguen en auge. El papel que desempeñan es fundamental para la mayoría de las industrias y para nuestro día a día: entretenimiento, comercio en línea, trámites con las administraciones públicas y un largo etcétera. A pesar de su relevancia, la seguridad de las aplicaciones web se pasa por alto.

Este reporte realizado por la Agencia de la Unión Europea para la Ciberseguridad (ENISA) [1] revela que muchas de las debilidades detectadas están relacionadas con la tecnología web. Este tipo de vulnerabilidades tienden a ser los objetivos principales de los ciberdelincuentes, ya que puede servir como punto de entrada para acceder a la red de la víctima. Este tipo de comportamiento está recogido en el *framework* MITRE ATT&CK como técnica T1190 [2].

Las empresas y las organizaciones no ignoran la ciberseguridad, para ellos es muy importante y la tienen en cuenta. Encuestas realizadas por ENISA indican que el gasto medio en seguridad de la información (SI) de los operadores de servicios esenciales y proveedores de servicios digitales en la Unión Europea fue de 700.000 euros en 2022, mientras que el gasto medio en 2023 fue de 5,1 millones de euros [3].

El principal problema que encuentran las industrias a la hora de adoptar medidas de ciberseguridad y planes ante ataques, es la falta de personal y talento. Esto se conoce como la *cybersecurity skill gap* [4]. Según el *Information Systems Audit and Control Association* (ISACA) [5], el estado actual de la ciberseguridad es desolador: una mano de obra envejecida junto con poca oferta de puestos de trabajo de nivel básico.

ISACA también menciona la existencia de barreras para los noveles que quieren dedicarse al sector (*gatekeeping*) y denuncia que las descripciones de los nuevos puestos de trabajo son problemáticos. Si esto se suma al agotamiento sufrido por los empleados y la incertidumbre económica actual, el resultado es la falta de profesionales dentro del sector de la ciberseguridad.

La estrategia de la Unión Europea para afrontar la *cyber skill gap* es fomentar programas e iniciativas centradas en formar a los estudiantes de enseñanza superior [6]. Estos programas se centran en solventar la falta de oferta educativa en ciberseguridad, la escasa interacción con la industria, la poca comprensión del mercado laboral, las plataformas desfasadas o poco realistas en los entornos educativos y las dificultades para seguir el ritmo del mundo exterior [7].

Sin embargo, aunque se resolviera la falta de profesionales en ciberseguridad, la responsabilidad de desarrollar software seguro y resiliente no deberían recaer únicamente en los expertos en ciberseguridad. Sigue existiendo la necesidad de formar a los desarrolladores tanto en desarrollo seguro como en *testing*, ya que juegan un papel crucial en detectar vulnerabilidades en el software y parchearlos.

La *cyber skill gap* también afecta a los programadores: la falta de formación y concienciación de los desarrolladores es la principal causa de las vulnerabilidades. Si no se centran los esfuerzos en enseñar a los desarrolladores cómo programar código seguro (sobretodo a los perfiles júnior), seguirán proliferando las vulnerabilidades en las aplicaciones web. Esto presenta un riesgo real de ser atacados por agentes externos malintencionados.

En el trabajo desarrollado por Gasiba T., Lechner U. y Pinto-Albuquerque M. [8], los autores proponen un juego basado en desafíos cuyo objetivo es la de concienciar y

formar a desarrolladores de software sobre cómo programar código seguro. Como resultado, los programadores que han participado respondieron positivamente ante la actividad.

Este trabajo propone el desarrollo de una aplicación web para la formación en desarrollo de software seguro, enfocada en la programación web. La propuesta consiste en una aplicación web que se ejecuta en la máquina del usuario, donde este pueda adquirir conocimientos sobre las vulnerabilidades más importantes de las aplicaciones web. Además, la aplicación presenta una serie de laboratorios que muestran dos escenarios: un escenario vulnerable, donde se simula una aplicación web con una serie de vulnerabilidades implementadas de forma intencional para que el desarrollador pueda poner en práctica sus habilidades como *pentester*; y un escenario seguro, donde las vulnerabilidades existentes en el escenario vulnerable están parcheadas y el usuario sea incapaz de realizar con éxito un ataque.

La presencia de estos dos escenarios seleccionables, junto con la presentación del código responsable de cada escenario y el material didáctico, tiene como objetivo desarrollar una doble visión al desarrollador: tanto de *pentester*, capaz de detectar vulnerabilidades y explotarlas mediante ataques, como de desarrollador de software seguro, capaz de solventar las debilidades y tapar los agujeros de seguridad presentes en el software que desarrolla.

Las contribuciones de este artículo se detallan a continuación:

1. El estudio y la comparativa de las distintas aplicaciones web vulnerables para la formación en seguridad y *pentesting* web.
2. La creación de una aplicación web para la formación en competencias en seguridad web con dos escenarios simultáneos: uno seguro y otro vulnerable. De esta forma, se proporcionan dos enfoques distintos al usuario: como desarrollador de software seguro y como *tester* de código. Además, la aplicación soporta varios lenguajes de programación populares que el usuario puede elegir: Java, Python y Node.js
3. La réplica de 7 vulnerabilidades que se encuentran dentro de las 5 categorías de los riesgos más habituales y destacables de la seguridad web.

II. Estado del arte

En la literatura, se han propuesto diversas plataformas y aplicaciones web que son diseñadas intencionadamente con vulnerabilidades. Por ejemplo, algunas contribuciones implementan cursos de seguridad web basados en el OWASP Top 10, destacando los desafíos de impartir este plan de estudios a través de la educación a distancia [9]. Otros trabajos proponen cursos universitarios de seguridad web con ejercicios prácticos para inculcar la *mentalidad de seguridad* [10]. Además, se han desarrollado herramientas específicas para la enseñanza de desarrollo web seguro en cursos de informática universitaria, utilizando entornos virtualizados [11].

Por otro lado, existe un enfoque distinto que se centra en la metodología de enseñanza práctica en ciberseguridad. Mukherjee y colaboradores revisan y modernizan los conocimientos existentes para ayudar a los educadores en el diseño de asignaturas de ciberseguridad [12]. Asimismo, Bratus S. et al. comparten su experiencia aplicando los principios del *Currículum Hacker* en programas de educación para estudiantes universitarios.

II-A. Vulnerabilidades comunes

A continuación se presentan los dos proyectos más relevantes cuyo objetivo es recoger las vulnerabilidades más comunes y significativas dentro del software y del desarrollo de aplicaciones web: el OWASP Top Ten y el CWE Top 25.

II-A1. OWASP Top Ten: El *OWASP Top Ten* [13] es un documento de concienciación sobre seguridad en aplicaciones web, enfocado hacia los desarrolladores y profesionales de la seguridad web. Este proyecto es creación de la fundación *Open Web Application Security Project* (OWASP) y se publica una nueva versión cada 3 o 4 años.

El objetivo de este documento es identificar las vulnerabilidades más comunes y relevantes de las aplicaciones web, así como establecer una clasificación de las 10 categorías de riesgos más importantes. Esto se basa en estadísticas proporcionadas por distintas organizaciones y encuestas a expertos en seguridad web. Cada categoría contiene información sobre los riesgos, incluyendo descripción, prevención y ejemplos, junto con referencias adicionales y los CWEs (Common Weakness Enumerations) que forman dicha categoría.

II-A2. MITRE CWE Top 25: El Common Weakness Enumeration (CWE) [14] es una lista de debilidades presentes en el software y en el hardware desarrollada por la comunidad. Esta lista ayuda a conocer y clasificar las debilidades en el software, firmware, hardware y componentes existentes que pueden contribuir a la introducción de vulnerabilidades.

Esta lista está mantenida por MITRE y el *Homeland Security Systems Engineering and Development Institute* (HSSEDI), y cuenta con un gran apoyo por parte de la comunidad que participa activamente en el desarrollo del proyecto. Esta lista se actualiza 3 o 4 veces por año y cuenta con algunas agrupaciones de debilidades más específicas, como por ejemplo el *Common Weakness Enumeration* (CWE) *Top 25 Most Dangerous Software Weaknesses list* (CWE Top 25) [15].

El CWE Top 25 es un subconjunto del la lista CWE, en donde se agrupan las 25 debilidades más peligrosas presentes en el software. Esta lista, que es actualizada cada año o dos años, se confecciona a partir de los datos proporcionados por los equipos del *Common Vulnerabilities and Exposures* (CVE), del *National Institute of Standards and Technology* (NIST), del *National Vulnerability Database* (NVD) y del *Common Vulnerability Scoring System* (CVSS).

II-A3. Comparativa OWASP Top Ten y CWE Top 25: Las dos listas presentadas previamente que recogen vulnerabilidades comunes, OWASP Top Ten y CWE Top 25, presentan las siguientes diferencias:

1. El CWE Top 25 recoge más vulnerabilidades (25) que el OWASP Top Ten (10). Eso sí, el OWASP Top Ten se compone de categorías de riesgo que agrupan varios CWEs.
2. El OWASP Top Ten se enfoca en los riesgos en aplicaciones web, mientras quel CWE Top 25 se centra en las debilidades en el software en general.
3. La frecuencia de actualización de las dos listas son distintas: mientras que OWASP renueva su lista cada 3 o 4 años, MITRE actualiza su *top* cada año o dos.

Tabla I
COMPARATIVA DE APLICACIONES WEB VULNERABLES

	DVWA	Webgoat	Juice Shop	Web Security Academy
Lenguaje	PHP	Java	Javascript (Node)	Desconocido
Número de vulnerabilidades	14	10	15	25
Tipos de vulnerabilidades	Vulnerabilidades web	Vulnerabilidades web	Vulnerabilidades web	Vulnerabilidades web
Enfoque	- Desarrollo seguro - Pentesting	- Desarrollo seguro - Pentesting	Pentesting	-Desarrollo seguro - Pentesting
Instalación/uso	- Instalación local - Docker container	- LAMP Stack - Docker container	- Node - Docker container - Proveedor de cloud (Google, AWS, Azure, etc.)	Registro online

4. A diferencia del CWE Top 25, el OWASP Top Ten no solo se centra en los datos. También agrega a la ecuación encuestas y votaciones realizadas a la comunidad de expertos en seguridad web para representar riesgos y vulnerabilidades que están en alza pero aún no aparecen en los datos.

En este artículo se va a centrar en el OWASP Top Ten por su enfoque hacia la seguridad web. Sin embargo, como se ha mencionado, las categorías del *top* contienen varios CWEs, muchos de los cuales están presentes en el CWE Top 25.

II-B. Aplicaciones vulnerables para la formación en seguridad web

En este subapartado se presentan 4 aplicaciones web vulnerables que se utilizan para la formación y concienciación en seguridad web. Las herramientas vulnerables estudiadas en este artículo son: DVWA, WebGoat, Juice Shop y Web Security Academy.

II-B1. DVWA: Damn Vulnerable Web Application (DVWA) [16] es una web desarrollada con PHP y MySQL que, de forma intencional, presenta vulnerabilidades. Está dirigida tanto a profesionales de la seguridad, para poner a prueba sus habilidades en un entorno legal, como a los desarrolladores de aplicaciones web, para que comprendan mejor los procesos de seguridad en el desarrollo web.

La aplicación presenta una interfaz web que contiene una serie de apartados que corresponden a las vulnerabilidades presentes en la aplicación. El objetivo de DVWA es permitir que el usuario practique ataques contra algunas de las vulnerabilidades web más comunes, permitiendo seleccionar la dificultad de los retos.

II-B2. WebGoat: WebGoat [17] es una aplicación web intencionadamente insegura mantenida por OWASP, diseñada para enseñar sobre seguridad en aplicaciones web. Está desarrollada en Java y ofrece un entorno interactivo compuesto por retos. El proyecto se basa en el OWASP Top Ten, mostrando los fallos de seguridad más comunes de las aplicaciones web en el lado del servidor.

II-B3. Juice Shop: OWASP también nos ofrece Juice Shop [18], una de las aplicaciones web inseguras más modernas, desarrollada en Angular y Node.js. A diferencia de las aplicaciones web vulnerables mencionadas anteriormente, Juice Shop se enfoca especialmente en los *pentesters* y expertos en ciberseguridad.

Esta aplicación está diseñada para ser utilizada en retos *Capture The Flag* (CTF), para realizar demostraciones y concienciar al público sobre la seguridad web, además de servir como entorno de pruebas para el desarrollo de herramientas de seguridad web. Juice Shop abarca todas las vulnerabilidades presentes en el OWASP Top Ten, así como otros fallos de seguridad comunes en aplicaciones del mundo real.

II-B4. Web Security Academy: PortSwigger Web Security Academy [19] es un recurso online gratuito para el entrenamiento y formación en seguridad web. Se actualiza constantemente y ofrece una amplia variedad de teoría y actividades interactivas sobre vulnerabilidades web. Además, incluye los *Learning Paths*, que son una recopilación de material enfocado en una temática específica.

Esta plataforma, también proporciona una certificación llamada *Burp Suite Certified Practitioner* (BSCP) para que los profesionales de la seguridad web puedan demostrar su competencia en este área.

II-B5. Comparativa de herramientas vulnerables: En la Tabla I se realiza una comparación de las aplicaciones web vulnerables para la formación de la seguridad web.

Se puede observar en la Tabla I que las aplicaciones web vulnerables se centran más en el perfil del *pentester* y de los profesionales de la seguridad que en el perfil del desarrollador web. Su contenido se enfoca en enseñar al usuario cómo llevar a cabo un ataque, dejando en segundo plano la enseñanza sobre cómo escribir código seguro.

Además, se nota la ausencia de la posibilidad de visualizar el código que se está ejecutando en los laboratorios en la mayoría de las aplicaciones analizadas. La única excepción es DVWA, que permite al usuario ver el código utilizado en la dificultad seleccionada mediante un botón en el menú.

Por último, es importante destacar que estas aplicaciones web se enfocan principalmente en simular entornos vulnerables para practicar ataques, pero no proporcionan una comparación directa con entornos seguros. Sería altamente beneficioso para los programadores poder distinguir entre el código seguro y el vulnerable. Esto les permitiría no solo aprender sobre las vulnerabilidades web y los ataques asociados, sino también comprender la diferencia entre trabajar en un entorno seguro y uno vulnerable. De esta manera, podrían desarrollar una perspectiva más completa, tanto desde la perspectiva de un *pentester* como de un desarrollador.

III. PROPUESTA DE APLICACIÓN

En base a las conclusiones obtenidas del estado del arte, este trabajo propone el desarrollo de una herramienta web vulnerable que aborde las deficiencias identificadas en las

Tabla II
CATEGORÍAS DEL OWASP TOP TEN

Categoría OWASP Top Ten	Tasa de incidencias	Explotación	Impacto
A01 Pérdida de Control de Acceso	55.97 %	6.92	5.93
A02 Fallos Criptográficos	46.44 %	7.29	6.81
A03 Inyección	19.09 %	7.25	7.15
A04 Diseño Inseguro	24.19 %	6.46	6.78
A05 Configuración de Seguridad Incorrecta	19.84 %	8.12	6.56
A06 Componentes Vulnerables y Desactualizados	27.96 %	5.00	5.00
A07 Fallos de Identificación y Autenticación	14.84 %	7.40	6.50
A08 Fallos en el Software y en la Integridad de los Datos	16.67 %	6.94	7.94
A09 Fallos en el Registro y la Monitorización	19.23 %	6.87	4.99
A10 Falsificación de Solicitudes del Lado del Servidor (SSRF)	2.72 %	8.28	6.72

aplicaciones web analizadas en la sección II-B5. Para ello, se ha diseñado una metodología para la selección de las vulnerabilidades a integrar en los laboratorios de aplicación web vulnerable, así como para determinar las funcionalidades a implementar en la aplicación, tomando como referencia las carencias observadas en las otras aplicaciones previamente estudiadas.

III-A. Selección de las vulnerabilidades

La selección de las vulnerabilidades a implementar en la aplicación se ha basado en la información proporcionada por el OWASP Top Ten[13]. Principalmente, se han estudiado las categorías que forman la lista y las estadísticas por cada categoría, como la tasa de incidencia, la puntuación ponderada de impacto y explotación siguiendo el CVSS v3.0 [20] y los CWEs [14] que forman cada categoría. Para la selección de las pruebas, se parte desde el OWASP Top Ten. Se seleccionan las categorías más relevantes para la primera versión de la aplicación y a partir de las categorías seleccionadas, se seleccionan los CWEs más importantes por cada categoría. A partir de esos CWEs, se implementan los laboratorios de la aplicación.

Como se puede observar en la Tabla II, cada categoría cuenta con una tasa de incidentes, una puntuación en explotación de la vulnerabilidad y una puntuación en impacto. La tasa de incidentes es el porcentaje de aplicaciones encuestadas por OWASP en el que, al menos, una vulnerabilidad de esa categoría se encuentra presente. Para calcular las puntuaciones en explotación e impacto, OWASP ha agrupado los CWEs relacionados por cada categoría y ha ponderado las puntuaciones de Explotación e Impacto según su puntuación CVSS 3.0. En la Tabla II se representa las puntuaciones del Top Ten, siguiendo la escala de colores definida en la sección *Qualitative Severity Rating Scale* del CVSS v3.0 [20].

A partir del Top Ten de OWASP, se han establecido una serie de requisitos para seleccionar las categorías más relevantes para el proyecto y descartar las categorías que no se ajusten bien al contenido de la aplicación. Los requisitos fijados para la selección de las categorías de la clasificación propuesta por OWASP son:

1. No debe ser necesario el uso de herramientas externas para poder explotar las vulnerabilidades presentes en la aplicación.
2. Las vulnerabilidades deben ser fáciles de implementar.
3. Las vulnerabilidades con mayor probabilidad de aparecer en una aplicación y las más relevantes para la formación del desarrollador deben estar presentes en la aplicación.
4. Las vulnerabilidades que sean más propensas a manifestarse en el código de los desarrolladores más inexpertos tienen que mostrarse en la aplicación.

Una vez establecidas las condiciones de selección, se escoge las siguientes categorías para la primera versión de la aplicación:

- A01:2021 – Pérdida de Control de Acceso
- A02:2021 – Fallos Criptográficos
- A03:2021 – Inyección
- A04:2021 – Diseño Inseguro
- A10:2021 – Falsificación de Solicitudes del Lado del Servidor (SSRF)

Las primeras 2 categorías (A01 y A02) cuentan con la mayor tasa de incidencias del Top Ten y presentan un riego medio y alto de explotación e impacto según la calificación CVSS. Las categorías A03 y A04 no superan en tasa de incidencias a las otras categorías que se encuentran por debajo, pero A03 y A04 son muy relevantes en la formación del desarrollador, ya que estas vulnerabilidades suelen aparecen en el código de los programadores júnior y sus puntuaciones en explotación e impacto son elevadas. La última categoría (A10), aunque cuenta con la menor tasa de incidencia de toda la lista, ocupa el primer lugar en la encuesta de la comunidad realizada por OWASP. Como se menciona en el OWASP Top Ten [13] «Los resultados de los datos se limitan principalmente a lo que podemos comprobar de forma automatizada. Si hablas con un profesional [...], te hablará de las cosas que encuentra y de las tendencias que ve y que aún no están en los datos. Se necesita tiempo para desarrollar metodologías de pruebas para ciertos tipos de vulnerabilidades y más tiempo aún para que esas pruebas sean automatizadas y ejecutadas contra una gran población de aplicaciones. Todo lo que encontramos está mirando al pasado y puede que falten algunas tendencias, que aún no están presentes en los datos».

Con las categorías de vulnerabilidades seleccionadas, se han implementado los laboratorios enunciados en la Tabla III seleccionando los CWEs más significativos de cada categoría.

III-B. Funciones de la aplicación

Como se ha mencionado previamente, las aplicaciones web vulnerables utilizadas para la formación en seguridad están más centradas en el entrenamiento del perfil de *pentester* que en la concienciación y la formación para el desarrollador.

Cabe destacar la ausencia del código que se está empleando cuando el usuario está realizando un laboratorio, a excepción

Tabla III
VULNERABILIDADES IMPLEMENTADAS

Ataque	CWE	Categoría OWASP Top 10
SQL Injection	**CWE-89**: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	A03 Inyección
Password Storing	**CWE-328**: Use of Weak Hash	A02 Fallos Criptográficos
	CWE-759: Use of a One-Way Hash without a Salt	
	CWE-312: Cleartext Storage of Sensitive Information	A04 Diseño Inseguro
Cross-Site Scripting (XSS)	**CWE-79**: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	A03 Inyección
Path Traversal	**CWE-22**: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	A01 Pérdida de Control de Acceso
Server-Side Request Forgery (SSRF)	**CWE-918**: Server-Side Request Forgery (SSRF)	A10 Falsificación de Solicitudes del Lado del Servidor (SSRF)
Insecure Direct Object References (IDOR)	**CWE-639**: Authorization Bypass Through User-Controlled Key	A01 Pérdida de Control de Acceso
Input Validation	**CWE-20**: Improper Input Validation	A03 Inyección

de la herramienta DVWA. Esta aplicación nos permite consultar el código en PHP responsable de la prueba que el usuario esté completando en ese momento.

Después del análisis de las aplicaciones web vulnerables para la formación en seguridad web del apartado II-B5, se ha decidido implementar las siguientes funcionalidades en nuestra solución de herramienta web vulnerable:

1. La implementación de una serie de pruebas (laboratorios) con las vulnerabilidades que se han seleccionado previamente. Los laboratorios estarán formados por dos escenarios: Un escenario vulnerable en el que el usuario sea capaz de realizar un ataque con éxito y un escenario seguro en el que el usuario no pueda ejecutar un ataque de forma exitosa.
2. En cada laboratorio se mostrará en pantalla el código que está involucrado. Si se ha seleccionado el escenario vulnerable, se expone el código vulnerable responsable de la prueba y viceversa.
3. El contenido de la aplicación estará dividido en secciones organizadas por las vulnerabilidades implementadas en la Tabla III. Por cada sección habrá contenido didáctico que explique la vulnerabilidad, cómo funcionan los distintos ataques, el impacto derivado de un ataque y las distintas medidas para subsanar la vulnerabilidad.
4. La aplicación contará con soporte para varios lenguajes de programación. El usuario podrá seleccionar el lenguaje en el que quiera formarse y realizar las pruebas propuestas.

Con las vulnerabilidades seleccionadas y las funciones de la aplicación definidas, se procede al diseño de la herramienta web vulnerable.

IV. DISEÑO Y ARQUITECTURA DE LA APLICACIÓN

Para el diseño de la aplicación web vulnerable se ha seguido la arquitectura más común en el desarrollo web, como se muestra en la Figura 1: Separar la aplicación en *front-end* y *back-end*. Estos componentes se comunicarán entre sí por HTTP mediante una *API Restful*.

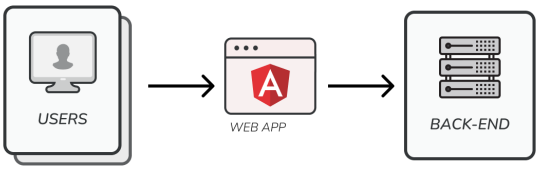

Figura 1. Arquitectura de la aplicación: Paso 0

El problema de esta arquitectura es la falta de soporte multilenguaje que se requiere para este proyecto: Aunque el *front-end* sea común para todos los usuarios, el *back-end* debería esta implementado en el lenguaje de programación que el usuario ha elegido dentro de la aplicación. Además, el usuario debe tener la opción de seleccionar otro lenguaje de programación mientras está utilizando la aplicación de forma instantánea, para poder comparar las diferentes implementaciones que se muestran en la página.

Para resolver este inconveniente, se ha añadido un *proxy* inverso en la arquitectura de la aplicación entre el *front-end* y el *back-end*, y se ha implementado varios *back-ends* con distintos lenguajes de programación con la misma definición de la API. Este diseño se representa en la Figura 2.

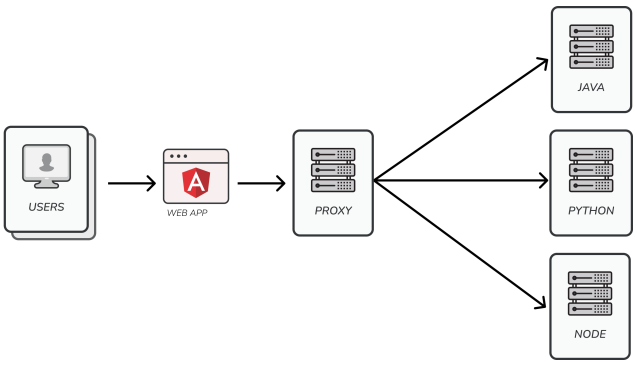

Figura 2. Arquitectura de la aplicación: Paso 1

Así, en la selección del lenguaje, el proxy se encarga de

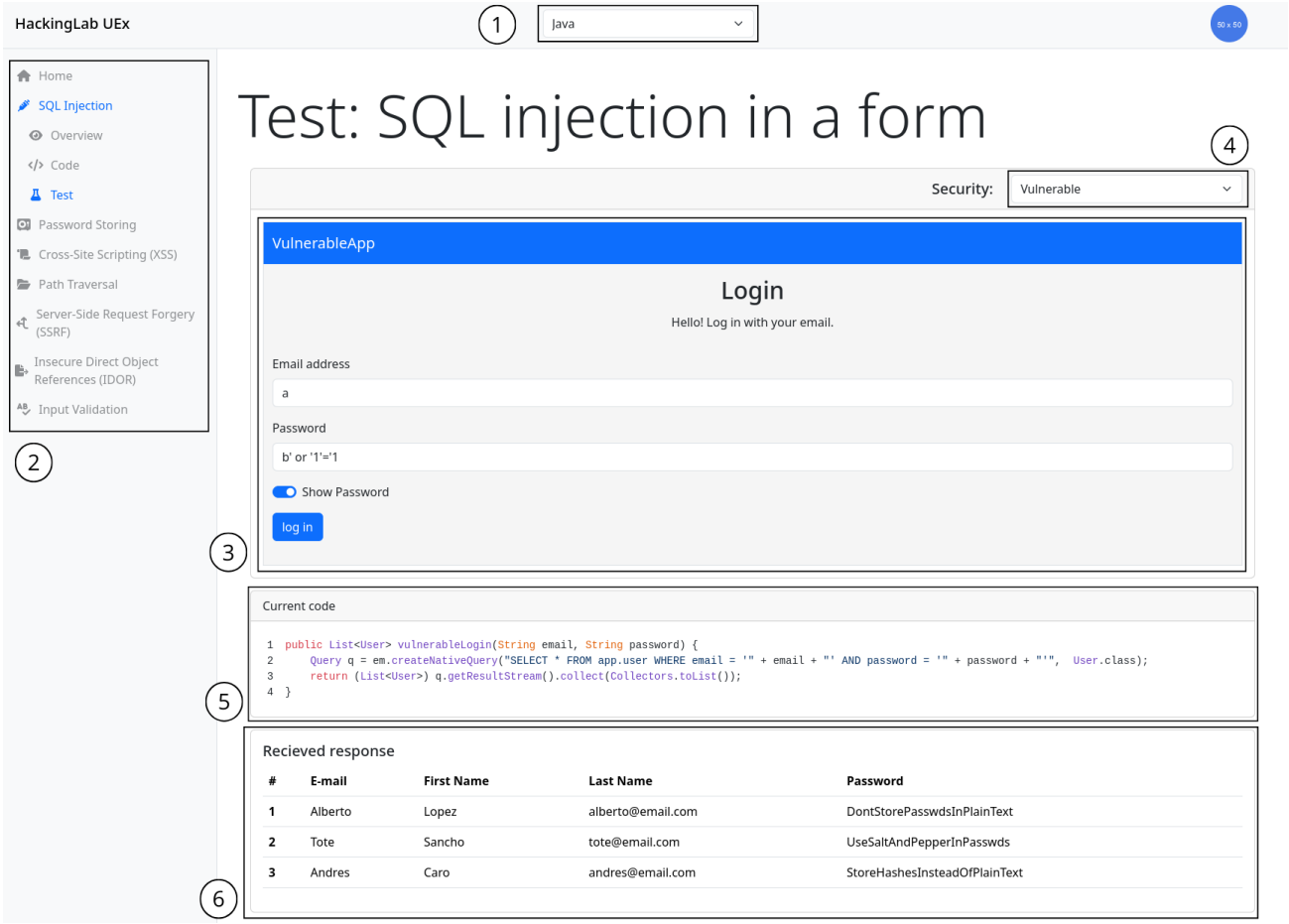

Figura 3. Pantalla de la aplicación: laboratorio SQL Injection

redirigir el tráfico al *back-end* seleccionado por el usuario, sin que el front-end tenga toda la responsabilidad de esta tarea. Además, como los 3 *back-ends* implementan la misma *API*, el *front-end* no tiene que modificar las solicitudes HTTP, independientemente del backend que reciba la solicitud.

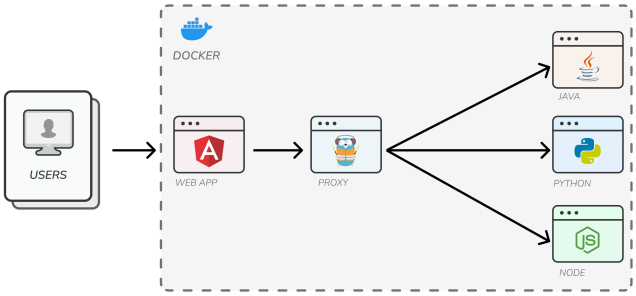

Figura 4. Arquitectura final de la aplicación

Para poder desplegar la aplicación en una misma máquina para que el usuario pueda utilizar el programa en local, cada componente está contenido en una imagen de Docker y el despliegue está definido en un archivo *docker-compose.yml*. El usuario simplemente desplegaría la aplicación utilizando el programa *docker compose* y ya tendría disponible la aplicación web en su máquina local. En la Figura 4 se describe la arquitectura final de la aplicación web vulnerable.

V. Resultados

En la Figura 3 se muestra la aplicación web vulnerable y los distintos componentes que forman el menú resaltados y enumerados. En la mencionada Figura 3, se aprecia que el selector de lenguaje (1) se encuentra en la barra superior de la aplicación. En esta versión, la aplicación soporta Java, Python y Node.js. Las secciones de la aplicación (2) las forman las vulnerabilidades que se seleccionaron en el apartado III-A, y cada sección está formada por tres apartados:

- **Overview**: En esta sección se proporciona una descripción general de la vulnerabilidad, se explican los mecanismos de los ataques, se discute el alcance del impacto y se detallan las medidas para detectar y prevenir la vulnerabilidad.
- **Code**: Este apartado comienza analizando el código vulnerable utilizado en la prueba. Posteriormente, se procede a implementar, paso a paso, diversas medidas y buenas prácticas para transformar el código en una versión segura, resistente a ataques.
- **Test**: En esta parte se realiza la implementación del laboratorio presentado en la Figura 3. Aquí se proporciona al usuario un entorno para llevar a cabo ataques contra la aplicación. Dependiendo del escenario seleccionado, el usuario puede tener éxito o no en sus intentos de ataque.

Observando la Figura 3, se puede examinar con mayor

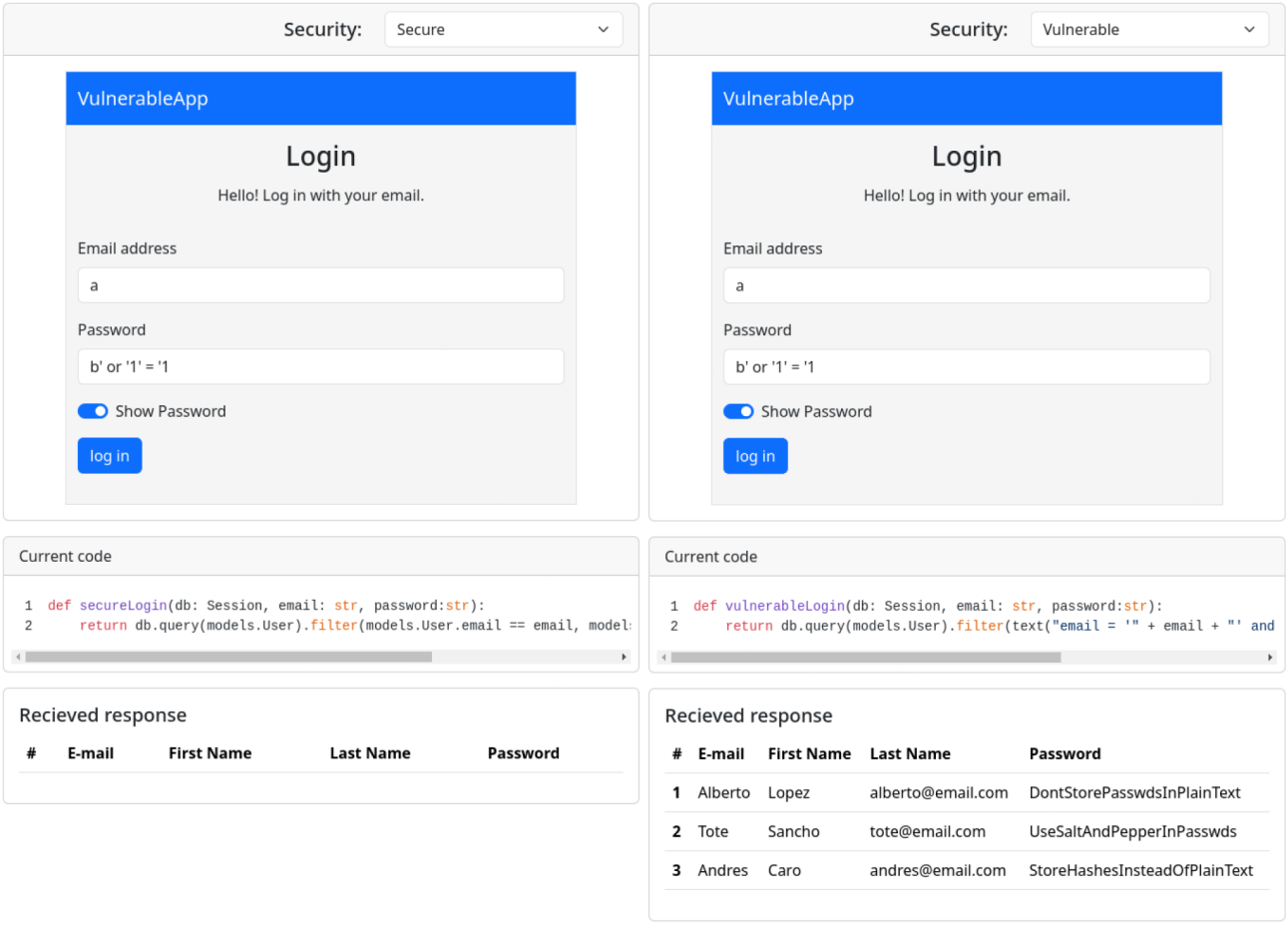

Figura 5. Comparación entre escenario vulnerable y seguro

detalle los laboratorios de la aplicación. Cada laboratorio está compuesto por una pantalla principal (3) que simula una aplicación web, ofreciendo al usuario la posibilidad de realizar ataques. Para elegir entre el escenario vulnerable y el seguro, la aplicación dispone de un selector (4).

Finalmente, debajo de la pantalla principal (5), se muestra el código correspondiente, ya sea vulnerable o seguro, dependiendo del escenario seleccionado. Además, el resultado del ataque (6) se presenta más abajo en la interfaz.

En la Figura 5 se expone los distintos resultados que arroja el escenario vulnerable y el escenario seguro ante el mismo ataque. En el laboratorio *SQL Injection* presentado también en la Figura 3, se realiza un ataque de inyección SQL en el formulario de *login* con el siguiente *payload*: «b' or '1' = '1».

El resultado del escenario vulnerable, representado a la derecha de la Figura 5, se muestra como el ataque ha finalizado de forma exitosa. Esto se expone en la sección *recieved response*, que muestra todos los usuarios existentes en la base de datos. En cambio, en la parte izquierda de la Figura 5, se observa como el ataque no ha sido satisfactorio en el escenario seguro. Comparando con la respuesta recibida en el escenario vulnerable, el escenario seguro no muestra ningún usuario existente de la base de datos.

Cabe destacar que el código que se muestra en los dos escenarios son diferentes, aunque en la Figura 5 no se pueda apreciar. en el Listado 1 se encuentra el código vulnerable del laboratorio *SQL Injection* y en el Listado 2 se encuentra el código seguro del mismo laboratorio. Ambos están programados en Python.

Observando el código, se puede descifrar el porqué de los resultados de cada escenario ante el mismo ataque. En el Listado 1, correspondiente al escenario vulnerable, vemos como los parámetros *email* y *password* son directamente concatenados en una cadena de caracteres que forma la cláusula *WHERE* de una sentencia SQL. Como el código no hace ningún tipo de validación, es posible introducir caracteres y palabras clave que el intérprete de consultas SQL interpretará como parte de la consulta legítima. De esta forma, con el *payload* «b' or '1' = '1», la cláusula *WHERE* quedaría de esta forma: «WHERE email = 'a' and password = 'b' or '1'='1'». Como la condición «'1'='1'» siempre se cumple para todas las filas de la tabla, la aplicación devuelve todas las filas que hay en la tabla.

En cambio, en el Listado 2, correspondiente al escenario seguro, vemos como se hace uso de una consulta parametrizada. De esta forma, el intérprete de consultas SQL puede diferenciar entre la consulta y los datos introducidos por el usuario. Así, evita la inyección de caracteres y palabras clave del lenguaje SQL dentro de la consulta.

```python
def vulnerableLogin(db: Session, email: str, password:str):
    return db.query(models.User)
        .filter(text("email = '" + email + "' and password = '" + password + "'"))
        .all()
```

Listado 1. Código vulnerable en Python

```python
def secureLogin(db: Session, email: str, password:str):
    return db.query(models.User)
        .filter(models.User.email == email, models.User.password == password)
        .all()
```

Listado 2. Código seguro en Python

VI. Conclusiones y trabajos futuros

Este trabajo propone una aplicación web vulnerable con un enfoque educativo dirigido específicamente a desarrolladores de software. La aplicación es multiplataforma y admite tres de los lenguajes de programación más populares: Java, Python y Node.js. Permite al usuario elegir entre dos escenarios simultáneos: uno seguro y otro vulnerable, ofreciendo así dos perspectivas diferentes: la de desarrollador de software seguro y la de *pentester*. Además, la aplicación ofrece la posibilidad de replicar siete ataques que se encuentran entre las cinco categorías más destacadas de riesgos de seguridad en las aplicaciones web.

Este proyecto sigue en desarrollo con el objetivo de ampliar las habilidades de seguridad de sus usuarios. Los esfuerzos se centran en replicar un mayor número de vulnerabilidades, admitir nuevos lenguajes de programación, crear un entorno de consulta *online* y aislado que garantice la seguridad, y desarrollar un panel de gamificación para analizar el progreso en las competencias adquiridas por los usuarios.

Agradecimientos

Cabe destacar que esta iniciativa se realiza en el marco de los fondos del Plan de Recuperación, Transformación y Resiliencia, financiados por la Unión Europea (Next Generation) en el marco del proyecto con referencia C109/23 «Proyecto estratégico UEx (Escuela Politécnica de Cáceres)-INCIBE».

Referencias

[1] E. U. A. for Cybersecurity, I. Lella, C. Ciobanu, E. Tsekmezoglou, M. Theocharidou, E. Magonara, A. Malatras, and R. Svetozarov Naydenov, *ENISA threat landscape 2023 – July 2022 to June 2023*, I. Lella, C. Ciobanu, M. Theocharidou, E. Magonara, A. Malatras, R. Svetozarov Naydenov, and E. Tsekmezoglou, Eds., 2023.

[2] "Exploit public-facing application, technique t1190 - enterprise — mitre att&ck®." [Online]. Available: https://attack.mitre.org/techniques/T1190/

[3] E. U. A. for Cybersecurity, M. Adamczyk, A. Drougkas, E. Philippou, P. Abel, F. Gratiolet, and E. Maaskant, *NIS investments – Cybersecurity policy assessment – November 2023*. European Union Agency for Cybersecurity, 2023.

[4] C. A. Ramezan, "Examining the cyber skills gap: An analysis of cybersecurity positions by sub-field," *Journal of Information Systems Education*, vol. 34, no. 1, pp. 94–105, 2023. [Online]. Available: Availableat:https://aisel.aisnet.org/jise/vol34/iss1/8

[5] ISACA, "State of cybersecurity 2023: Global update on workforce efforts, resources and cyberoperations state of cybersecurity," 2023.

[6] E. U. A. for Cybersecurity, J. R. Nurse, K. Adamos, A. Grammatopoulos, and F. Di Franco, *Addressing the EU cybersecurity skills shortage and gap through higher education*, 2021.

[7] E. U. A. for Cybersecurity, T. De Zan, and F. Di Franco, *Cybersecurity skills development in the EU – The certification of cybersecurity degrees and ENISA's Higher Education Database*. European Network and Information Security Agency, 2019.

[8] T. Gasiba, U. Lechner, and M. Pinto-Albuquerque, "Cybersecurity challenges for software developer awareness training in industrial environments," in *Innovation Through Information Systems*, F. Ahlemann, R. Schütte, and S. Stieglitz, Eds. Cham: Springer International Publishing, 2021, pp. 370–387.

[9] B. Ksiezopolski, K. Mazur, M. Miskiewicz, and D. Rusinek, "Teaching a hands-on ctf-based web application security course," *Electronics (Switzerland)*, vol. 11, 11 2022.

[10] T. Srivatanakul and T. Moore, "Promoting security mindset through hands-on exercises for computer science undergraduate students," in *2021 2nd Information Communication Technologies Conference (ICTC)*, 2021, pp. 343–347.

[11] L.-C. Chen and L. Tao, "Teaching web security using portable virtual labs," in *2011 IEEE 11th International Conference on Advanced Learning Technologies*, 2011, pp. 491–495.

[12] M. Mukherjee, N. T. Le, Y.-W. Chow, and W. Susilo, "Strategic approaches to cybersecurity learning: A study of educational models and outcomes," *Information*, vol. 15, p. 117, 2 2024.

[13] "Owasp top 10:2021." [Online]. Available: https://owasp.org/Top10/

[14] "Cwe - common weakness enumeration." [Online]. Available: https://cwe.mitre.org/about/index.html

[15] "Cwe - cwe top 25 most dangerous software weaknesses." [Online]. Available: https://cwe.mitre.org/top25/

[16] "Github - digininja/dvwa: Damn vulnerable web application (dvwa)." [Online]. Available: https://github.com/digininja/DVWA

[17] "Github - webgoat/webgoat: Webgoat is a deliberately insecure application." [Online]. Available: https://github.com/WebGoat/WebGoat

[18] "Github - juice-shop/juice-shop: Owasp juice shop: Probably the most modern and sophisticated insecure web application." [Online]. Available: https://github.com/juice-shop/juice-shop

[19] "Web security academy: Free online training from portswigger." [Online]. Available: https://portswigger.net/web-security

[20] "Cvss v3.0 specification document." [Online]. Available: https://www.first.org/cvss/v3.0/specification-document