Seguridad en Aplicaciones Web mediante la Implementación de Múltiples Estrategias Defensivas

Ryan E. Vega Torres
Maestría en Ciencias de Computadoras
Mentor: Jeffrey Duffany, Ph.D.
Universidad Politécnica de Puerto Rico
Graduate Project EXPO, Mayo 2026

Abstracto — *Este trabajo presenta una evaluación comparativa de vulnerabilidades en aplicaciones web realizada en un entorno de laboratorio controlado con Kali Linux. Para el análisis se utilizaron OWASP Juice Shop, Damn Vulnerable Web Application (DVWA) y OWASP Mutillidae II. Las pruebas incluyeron encabezados de seguridad, inyección SQL, secuencias de comandos en sitios cruzados (XSS), autenticación por fuerza bruta, inyección de comandos y exposición de directorios o endpoints. La evaluación combinó herramientas manuales y automatizadas como OWASP ZAP, Burp Suite, SQLMap, XSStrike, Wfuzz, Gobuster, Dirsearch y comandos de validación en terminal. El enfoque permitió comparar líneas base iniciales con escenarios posteriores a mitigaciones, configuraciones más restrictivas o ajustes en los niveles de seguridad. Los resultados muestran que el uso combinado de herramientas ofrece una visión más completa del comportamiento de cada aplicación y que la defensa en profundidad puede reducir la exposición, aunque no sustituye la corrección de vulnerabilidades internas.*

Palabras Clave — *Entorno de Laboratorio Controlado, Evaluación Comparativa de Vulnerabilidades, Pruebas de Penetración, Seguridad en Aplicaciones Web.*

INTRODUCCIÓN

La seguridad de los sistemas web continúa siendo un tema importante debido al aumento de amenazas dirigidas a servicios conectados a Internet. En este contexto, las herramientas de análisis dinámico son útiles porque permiten evaluar una aplicación mientras está en ejecución, sin necesidad de revisar directamente su código fuente. Sin embargo, sus resultados pueden variar según la arquitectura, configuración y comportamiento de cada aplicación, por lo que combinar métodos manuales y automatizados puede ofrecer una evaluación más completa que depender de una sola herramienta [1].

Las pruebas de penetración en aplicaciones web permiten identificar vulnerabilidades antes de que puedan ser explotadas y ayudan a evaluar si los controles existentes son suficientes para reducir el riesgo. Estas pruebas son especialmente relevantes cuando las aplicaciones presentan validaciones débiles, configuraciones inseguras o controles defensivos incompletos, ya que estas condiciones pueden afectar la confidencialidad, integridad y disponibilidad de la información [2].

Este proyecto se enfoca en la evaluación de vulnerabilidades comunes en aplicaciones web dentro de un entorno de laboratorio controlado. El análisis consideró distintos tipos de fallas asociadas con la validación de entradas, autenticación, configuración del servidor, exposición de recursos y controles de protección del navegador. De esta manera, fue posible observar cómo diferentes condiciones técnicas pueden influir en el nivel de exposición de una aplicación y en la forma en que las herramientas detectan o interpretan los hallazgos [3] [4].

Este trabajo presenta una evaluación comparativa de vulnerabilidades utilizando Kali Linux y aplicaciones web intencionalmente vulnerables. El propósito principal fue comparar el comportamiento de las aplicaciones evaluadas y observar cómo cambiaban los resultados ante mitigaciones, configuraciones más restrictivas o ajustes en los niveles de seguridad.

Marco Conceptual y Revisión de Literatura

La seguridad en aplicaciones web busca reducir riesgos que afecten la confidencialidad, integridad y disponibilidad de los sistemas. Las pruebas de penetración permiten evaluar aplicaciones mediante escenarios controlados para detectar vulnerabilidades y validar controles existentes [1]. El análisis dinámico complementa este proceso al observar la aplicación en ejecución, aunque sus resultados pueden variar según la arquitectura, autenticación, rastreo y configuración de cada herramienta [2]. Por ello, combinar pruebas manuales y automatizadas permite interpretar mejor los hallazgos y confirmar la evidencia obtenida.

Vulnerabilidades Comunes en Aplicaciones Web

Las vulnerabilidades evaluadas en este proyecto representan distintas formas en que una aplicación web puede quedar expuesta ante entradas mal controladas, configuraciones débiles o recursos accesibles innecesariamente. Entre ellas se incluyeron la inyección SQL, las secuencias de comandos en sitios cruzados (XSS), la autenticación por fuerza bruta, la inyección de comandos, la ausencia o debilidad de encabezados de seguridad y la exposición de endpoints o directorios. Estas categorías permitieron analizar la seguridad desde varias capas, incluyendo la validación de entradas, el control de acceso, la configuración del servidor y la visibilidad de recursos [1].

La inyección SQL y XSS se relacionan principalmente con el manejo inseguro de datos enviados por el usuario, mientras que la inyección de comandos representa un riesgo más directo sobre el servidor porque puede permitir la ejecución de instrucciones no autorizadas en el sistema operativo [4]. Por otro lado, la autenticación por fuerza bruta, la exposición de directorios y los encabezados de seguridad ausentes o mal configurados reflejan debilidades que pueden ampliar la superficie de ataque o reducir la protección disponible para el navegador, las sesiones y los recursos de la aplicación.

Metodología de Pruebas de Seguridad

Esta sección describe el enfoque utilizado para realizar las pruebas de seguridad en un entorno controlado. Se presentan el laboratorio, las aplicaciones evaluadas y la razón por la cual fueron seleccionadas para comparar vulnerabilidades, herramientas y comportamientos técnicos.

Entorno de Laboratorio

El proyecto se desarrolló en un laboratorio controlado para ejecutar las pruebas de forma segura y sin afectar sistemas de producción. Se utilizó Kali Linux dentro de Oracle VirtualBox como ambiente principal, lo que permitió organizar las pruebas, ejecutar herramientas manuales y automatizadas, y manejar las aplicaciones vulnerables de manera controlada.

Aplicaciones Evaluadas

Para el proyecto se seleccionaron tres aplicaciones web intencionalmente vulnerables: OWASP Juice Shop, Damn Vulnerable Web Application (DVWA) y OWASP Mutillidae II. Estas permitieron comparar vulnerabilidades desde distintas arquitecturas, niveles de seguridad y comportamientos técnicos. Juice Shop representó una aplicación moderna y dinámica; DVWA permitió evaluar SQL Injection, XSS y fuerza bruta por niveles de seguridad; y Mutillidae II complementó el análisis con inyección de comandos y enumeración de directorios. Inicialmente se consideró WebGoat, pero fue sustituido por Mutillidae II por ajustarse mejor al enfoque comparativo del proyecto.

Análisis y Discusión de Resultados

Esta sección presenta los hallazgos obtenidos durante las pruebas de seguridad y su análisis comparativo entre las aplicaciones evaluadas. Se examina el comportamiento observado antes y después de mitigaciones, configuraciones restrictivas o cambios en los niveles de seguridad, con énfasis en la efectividad de los controles aplicados.

Encabezados de Seguridad

La evaluación inicial de OWASP Juice Shop evidenció una protección parcial en los encabezados HTTP antes de aplicar la configuración defensiva en Nginx. La validación mostrada en la Ilustración 1, realizada con curl -I sobre http://localhost:3000, confirmó la presencia de encabezados predeterminados como X-Content-Type-Options: nosniff y X-Frame-Options: SAMEORIGIN, pero sin controles adicionales como Content-Security-Policy, Strict-Transport-Security, Referrer-Policy y Permissions-Policy.

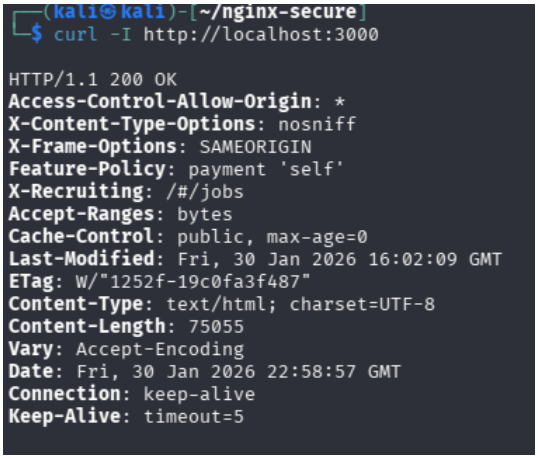

Ilustración 1
Respuesta HTTP Inicial de OWASP Juice Shop antes de Aplicar la Mitigación de Encabezados de Seguridad

La mitigación se implementó mediante un reverse proxy con Nginx utilizando directivas add_header para reforzar la respuesta HTTP enviada al navegador. Como se muestra en la Ilustración 2, se incorporaron controles como Content-Security-Policy, X-Frame-Options: DENY, X-Content-Type-Options: nosniff, Referrer-Policy, Permissions-Policy, Cache-Control y Pragma, añadiendo una capa defensiva sin modificar directamente el código fuente de la aplicación.

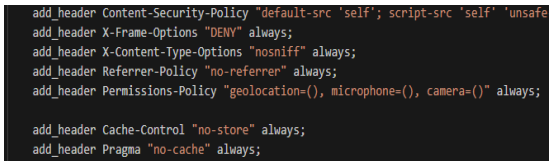

Ilustración 2
Fragmento de Configuración en Nginx para la Incorporación de Encabezados HTTP Defensivos

La validación posterior a la mitigación confirmó que la respuesta HTTP ya incluía los encabezados defensivos configurados en Nginx. La Ilustración 3 evidencia la incorporación de controles como Content-Security-Policy, Referrer-Policy, Permissions-Policy y Strict-Transport-Security, junto con el fortalecimiento de X-Frame-Options mediante el valor DENY. Este resultado permitió comprobar que el reverse proxy estaba aplicando correctamente la capa de protección definida para reducir la exposición inicial de la aplicación.

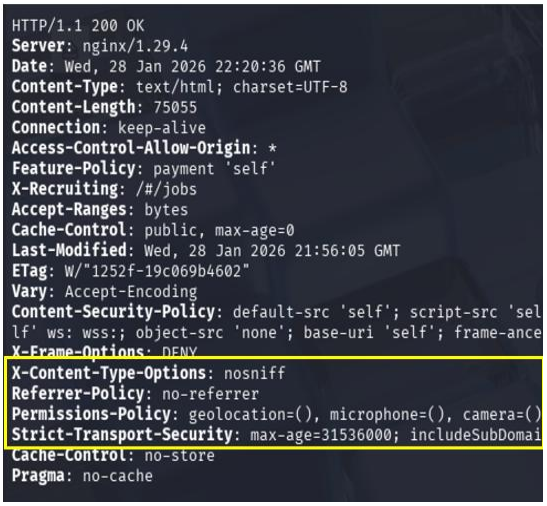

Ilustración 3
Respuesta HTTP Posterior a la Mitigación mostrando los Encabezados de Seguridad Añadidos mediante Nginx

Inyección SQL

Durante la prueba de SQL Injection en DVWA, Burp Suite se utilizó para interceptar la solicitud HTTP y verificar el parámetro evaluado antes de ejecutar SQLMap. La Ilustración 4 recoge esta etapa inicial del procedimiento, donde se observa la solicitud capturada junto con el contexto del módulo vulnerable en la aplicación. Las cookies de sesión obtenidas durante la intercepción permitieron conservar el contexto correspondiente a cada nivel de seguridad durante la validación automatizada. Para evitar evidencia repetitiva, se seleccionaron dos escenarios representativos: el nivel Low, en el que la exposición fue confirmada, y el nivel High, donde la explotación no fue confirmada bajo condiciones más restrictivas.

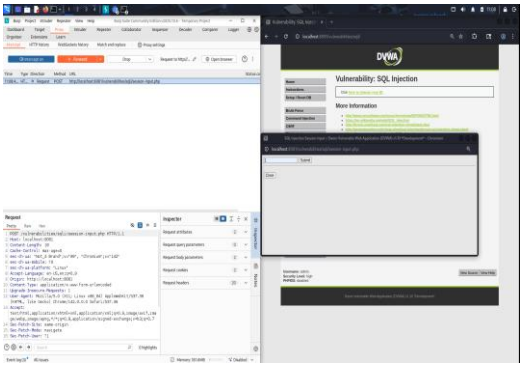

Ilustración 4
Solicitud HTTP de DVWA Interceptada en Burp Suite durante la Prueba de SQL Injection

La Tabla 1 resume los valores de sesión identificados en las solicitudes HTTP capturadas con Burp Suite mientras se interactuaba con DVWA desde la aplicación web. Estos valores, incluyendo PHPSESSID y el parámetro security, fueron utilizados para ejecutar SQLMap manteniendo el contexto correspondiente a cada nivel de seguridad. De esta manera, la validación automatizada pudo realizarse sobre sesiones activas y configuraciones específicas, evitando que los resultados se mezclaran entre los niveles evaluados.

Tabla 1
Cookies de Sesión Utilizadas para Mantener el Contexto de Prueba en SQLMap

DVWA Security Level	Session Cookie Used
Low	PHPSESSID=ibmuf675dlu3harvfvqrqiqio4; security=low
High	PHPSESSID=ve953dpnhu6h262uvaainnr8t7; security=high

Los resultados obtenidos con SQLMap mostraron una diferencia clara entre el nivel Low y el nivel High de DVWA. En el nivel Low, la herramienta identificó con mayor facilidad la exposición a SQL Injection, ya que la aplicación mantenía controles mínimos sobre el parámetro evaluado. La Ilustración 5 recoge esta evidencia al mostrar cómo SQLMap reconoce el parámetro id y lo asocia con un comportamiento vulnerable durante la prueba. Este resultado indica que la solicitud procesada por la aplicación podía ser manipulada de forma insegura y que la vulnerabilidad pudo ser validada con menor dificultad en este nivel de seguridad.

Ilustración 5
Validación de SQL Injection en DVWA Nivel Low mediante SQLMap

En contraste, el nivel High presentó un comportamiento más restrictivo durante la validación automatizada. Aunque se utilizó el mismo enfoque, primero capturando la solicitud HTTP con Burp Suite y luego ejecutándola en SQLMap con las cookies de sesión correspondientes, la herramienta no logró confirmar una explotación exitosa. La Ilustración 6 refleja este resultado al mostrar que SQLMap continuó realizando pruebas sobre el parámetro id, pero sin identificar una inyección de forma concluyente bajo ese nivel de seguridad. Este comportamiento no representa una falla del procedimiento, sino una evidencia de que los controles aplicados en DVWA High aumentaron la dificultad de explotación y redujeron la capacidad de confirmación automatizada.

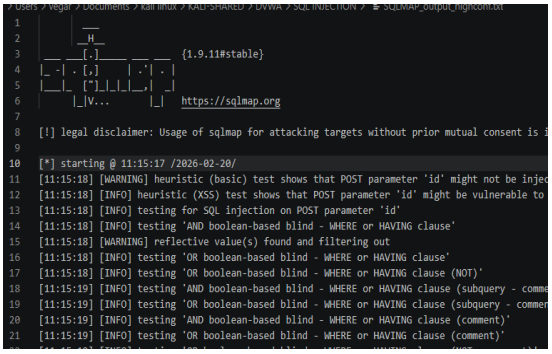

Ilustración 6
Resultado de SQLMap en DVWA Nivel High bajo Controles más Restrictivos

Cross Site Scripting (XSS)

La prueba de XSS comenzó con la interacción directa en el módulo Reflected XSS de DVWA. Durante esta etapa, Burp Suite permitió interceptar la solicitud HTTP, identificar el parámetro name como punto de entrada y confirmar el contexto de sesión utilizado en los niveles Low y High. La Ilustración 7 documenta esta fase inicial, donde se aprecia la solicitud capturada desde la aplicación antes de continuar con la validación automatizada. Posteriormente, XSStrike utilizó esta información para evaluar el comportamiento de la aplicación y comparar las diferencias entre una configuración con controles mínimos y otra con controles más restrictivos.

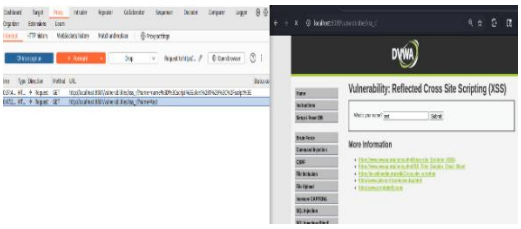

Ilustración 7
Solicitud del Módulo Reflected XSS de DVWA Capturada en Burp Suite

En el nivel Low, XSStrike pudo ejecutarse de manera más directa porque la solicitud requería menos parámetros adicionales y el campo "name" podía evaluarse con menos restricciones. La Ilustración 8 registra la ejecución de la herramienta utilizando la cookie de sesión capturada en DVWA, lo que permitió mantener el contexto correcto del nivel low durante la prueba automatizada. En cambio, para el nivel Impossible fue necesario construir una solicitud más detallada, incorporando el user_token, la cookie de sesión y el valor security=impossible, como se documenta en la Ilustración 9.

```
┌──(xsenv)─(kali㉿kali)-[~/XSStrike]
└─$ python xsstrike.py -u "http://localhost:8081/vulnerabilities/xss_r/?name=test1" --headers "Cookie: PHPSESSID=4k3fnsoc66t4sfkmamp57fl4n4; security=low" --fuzzer --threads 5 --console-log-level INFO > /media/sf_KALI-SHARED/DVWA/xss_low_results.txt
```

Ilustración 8
Ejecución de XSStrike en DVWA Nivel Low utilizando la Cookie de Sesión Capturada

```
┌──(xsenv)─(kali㉿kali)-[~/XSStrike]
└─$ python xsstrike.py \
-u "http://localhost:8081/vulnerabilities/xss_r/?name=&user_token=11bd54ae7b29dd3fef8ee75939a8fa0a" \
--headers "Cookie: PHPSESSID=4k3fnsoc66t4sfkmamp57fl4n4; security=impossible" \
--fuzzer \
--threads 5 \
--console-log-level INFO \
> /media/sf_KALI-SHARED/DVWA/xss_impossible_results.txt
```

Ilustración 9
Ejecución de XSStrike en DVWA Nivel Impossible utilizando Token y Cookie de Sesión

La comparación entre los resultados de XSStrike en DVWA Low e Impossible mostró una diferencia clara en el manejo de entradas asociadas con XSS reflejado. En el nivel Low, la herramienta reportó múltiples payloads como passed, incluyendo pruebas con etiquetas básicas, variaciones de script, manejadores de eventos, SVG, null byte e inyecciones por atributos. La Ilustración 10 respalda este hallazgo al presentar la salida de XSStrike para el nivel Low, donde se observa que el parámetro name fue evaluado con controles mínimos. En cambio, en el nivel Impossible, los mismos tipos de payloads fueron reportados como filtered, sin observarse una reflexión explotable, según se documenta en la Ilustración 11. Además, este escenario requirió un contexto más específico, incluyendo el uso del parámetro user_token, lo que demuestra que los controles más restrictivos hicieron más difícil la validación automatizada.

```
$ cat /media/sf_KALI-SHARED/DVWA/xss_low_results.txt

XSStrike v3.1.5

[~] WAF Status: Offline
[!] Fuzzing parameter: name
[+] [passed]   <test
[+] [passed]   <test//
[+] [passed]   <test>
[+] [passed]   <test x>
[+] [passed]   <test x=y
[+] [passed]   <test x=y//
[+] [passed]   <test/oNxX=yYy//
[+] [passed]   <test oNxX=yYy>
[+] [passed]   <test onload=x
```

Ilustración 10
Resultados de XSStrike en DVWA Nivel Low

```
[~] WAF Status: Offline
[!] Fuzzing parameter: name
[-] [filtered] <test
[-] [filtered] <test//
[-] [filtered] <test>
[-] [filtered] <test x>
[-] [filtered] <test x=y
[-] [filtered] <test x=y//
[-] [filtered] <test/oNxX=yYy//
[-] [filtered] <test oNxX=yYy>
[-] [filtered] <test onload=x
[-] [filtered] <test/o%00nload=x
[-] [filtered] <test sRc=xxx
[-] [filtered] <test data=asa
```

Ilustración 11
Resultados de XSStrike en DVWA Nivel Impossible

Enumeración de Directorios

La evaluación de enumeración de directorios comenzó en el módulo Directory Browsing de OWASP Mutillidae II bajo Security Level 0. La Ilustración 12 introduce el escenario utilizado para esta prueba, donde la propia aplicación presenta un módulo orientado a explorar el comportamiento de navegación y acceso a directorios. Antes de ejecutar herramientas automatizadas, se realizó una validación manual con Burp Suite, enviando la solicitud a Repeater y modificando rutas o parámetros para observar la respuesta del servidor.

Ilustración 12
Módulo Directory Browsing de OWASP Mutillidae II

Luego de la validación manual con Burp Suite, se ejecutó Dirsearch contra OWASP Mutillidae II para ampliar la búsqueda de rutas y recursos visibles. La Ilustración 13 presenta el resumen del escaneo inicial contra http://127.0.0.1/, donde se identificaron diferentes respuestas del servidor, incluyendo recursos accesibles, redirecciones, accesos no autorizados, rutas prohibidas y errores internos.

Después del escaneo inicial, se aplicó una mitigación mediante un reverse proxy con el propósito de limitar el acceso a rutas sensibles expuestas en la aplicación. La Ilustración 14 presenta el fragmento de configuración utilizado para bloquear directorios o archivos como /.git, /phpmyadmin, /phpinfo.php, /installation.php y /passwords, mientras el tráfico permitido continuaba siendo redirigido hacia OWASP Mutillidae II. Esta configuración permitió crear un escenario más restringido antes de repetir la enumeración con Dirsearch y evaluar si la exposición observada en la línea base inicial se reducía después de la mitigación.

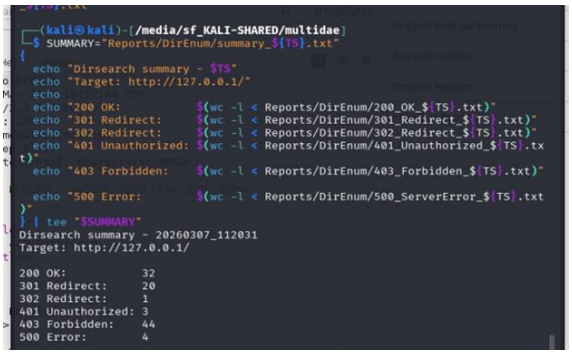

Ilustración 13
Resumen Inicial de Dirsearch en OWASP Mutillidae II antes de la Mitigación

```
    location / {
    proxy_pass http://127.0.0.1:80;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto https;
}

# Block exposed Git repositories
location ~ /\.git {
    deny all;
}

# Block phpMyAdmin
location /phpmyadmin {
    deny all;
}

# Block phpinfo file
location = /phpinfo.php {
    deny all;
}

# Block installation script
location = /installation.php {
    deny all;
}

# Block passwords directory
location /passwords {
    deny all;
}
```

Ilustración 14
Fragmento de Configuración del Reverse Proxy utilizado para Bloquear Rutas Sensibles

Después de aplicar las reglas de filtrado en el reverse proxy, se repitió la enumeración con Dirsearch contra https://127.0.0.1/. La Ilustración 15 registra el resumen posterior a la mitigación, donde se observa una enumeración más restrictiva en comparación con la línea base inicial. En particular, se redujeron respuestas asociadas con redirecciones, accesos no autorizados y errores internos, mientras algunos recursos públicos continuaron accesibles.

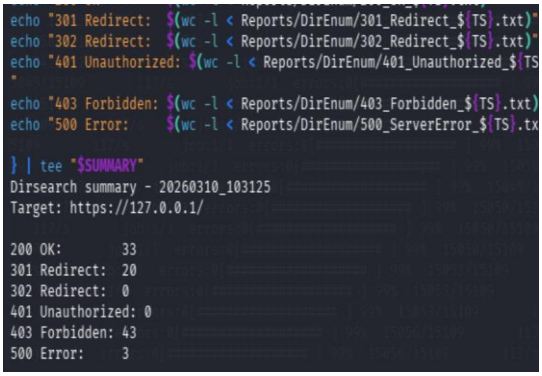

Ilustración 15
Resumen de Dirsearch Posterior a la Mitigación mediante Reverse Proxy en OWASP Mutillidae II

Robustez de Autenticación y Resistencia a Intentos Automatizados

La prueba de autenticación comparó dos escenarios en OWASP Juice Shop: una cuenta creada con una contraseña débil basada en información personal del perfil y otra cuenta configurada con una contraseña más compleja. La Ilustración 16 establece el punto de partida de esta prueba, mostrando la cuenta creada dentro de la aplicación antes de ejecutar los intentos automatizados. Para generar las combinaciones de prueba, se utilizaron datos como nombre, ciudad, mascota, año de nacimiento y patrón de correo electrónico. Luego, esas combinaciones fueron probadas contra el endpoint de autenticación para observar si alguna coincidía con la contraseña real y determinar la diferencia de resistencia entre una credencial predecible y una credencial más robusta.

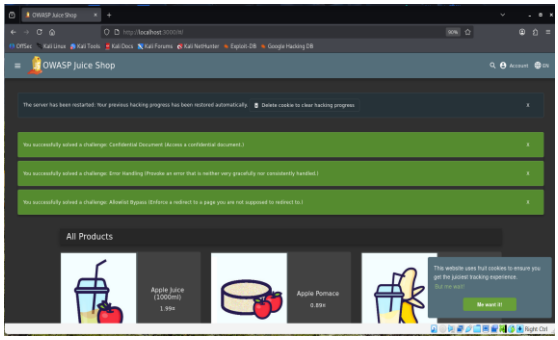

Ilustración 16
Cuenta creada en OWASP Juice Shop para la Prueba de Autenticación

La Tabla 2 resume los datos utilizados para crear dos cuentas de prueba en OWASP Juice Shop. La Cuenta 1 utilizó una contraseña basada directamente en información personal del perfil, mientras que la Cuenta 2 utilizó una contraseña más compleja para comparar la resistencia frente a intentos automatizados.

Tabla 2
Información de Perfiles utilizada para Crear Patrones de Contraseña en OWASP Juice Shop

Account 1	Account 2
Name: Carlos Rivera Town: Bayamón Pet: Rocky Birth Year: 1998 Email pattern:CarlosR98 **PASSWORD:rocky1998**	Name: Laura Gómez Town: Ponce Pet: Luna Birth Year: 1995 Favorite Color: Blue Hobby: Photography **PASSWORD: L9$uN!_72pQ**

El script automatizado probó una lista de contraseñas generadas a partir de información personal del perfil, incluyendo nombre, ciudad, mascota y año de nacimiento. La Ilustración 17 documenta la ejecución de la prueba contra el endpoint de autenticación de OWASP Juice Shop, donde la combinación rocky1998 fue identificada como válida. Este resultado confirma que una contraseña basada directamente en datos personales puede ser descubierta con mayor facilidad mediante intentos automatizados, especialmente cuando el atacante puede construir patrones predecibles a partir de información asociada al usuario.

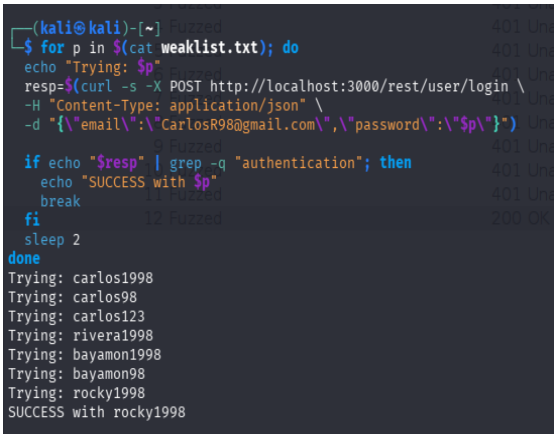

Ilustración 17
Validación Automatizada de una Credencial Predecible en OWASP Juice Shop

El script también probó combinaciones generadas a partir de la información del perfil de la Cuenta 2, incluyendo nombre, ciudad, mascota y año de nacimiento. A diferencia de la cuenta con contraseña débil, la Ilustración 18 registra múltiples intentos automatizados sin que se produjera un mensaje de éxito. Este resultado evidenció una mayor resistencia frente a contraseñas generadas con patrones personales predecibles, ya que la credencial real no seguía una estructura fácilmente deducible a partir de los datos del perfil.

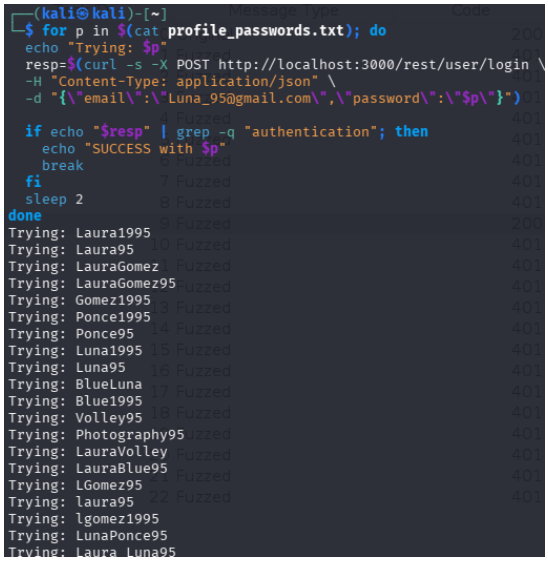

Ilustración 18
Intentos Automatizados contra una Cuenta con Contraseña más Robusta en OWASP Juice Shop

Autenticación por Fuerza Bruta

La prueba de autenticación por fuerza bruta se realizó en el módulo Brute Force de DVWA. La Ilustración 19 documenta la validación manual inicial, donde la solicitud generada desde la aplicación fue capturada con Burp Suite para analizar el comportamiento del servidor. Posteriormente, la solicitud se envió a Repeater con el fin de modificar credenciales y observar cambios en el código HTTP, los mensajes de error, las redirecciones y el tamaño de la respuesta. En este proceso, el Content-Length resultó especialmente útil para distinguir entre intentos fallidos y exitosos, sobre todo en niveles donde el código HTTP permanecía sin cambios. A partir de esta base manual, los resultados automatizados permitieron comparar cómo variaba el comportamiento de la aplicación entre los distintos niveles de seguridad.

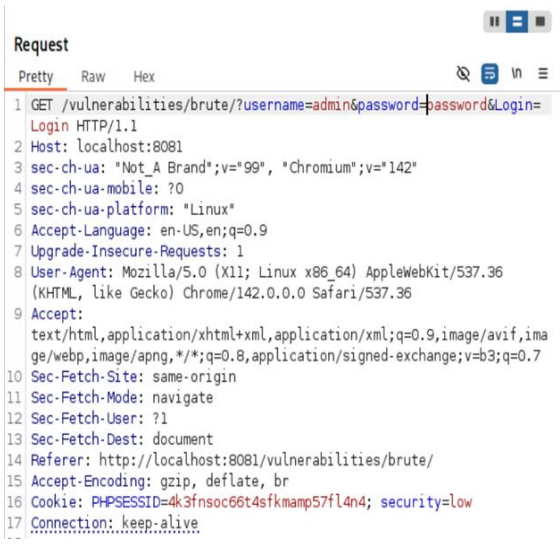

Ilustración 19
Módulo Brute Force de DVWA Capturado durante la Validación Manual en Burp Suite

Después de validar manualmente la solicitud, se utilizaron Wfuzz y el comando curl para automatizar intentos de autenticación contra el mismo endpoint de DVWA. Para mantener el contexto correcto de la prueba, se incluyó la cookie de sesión correspondiente junto con el valor del nivel de seguridad evaluado. La Ilustración 20 documenta el uso de Wfuzz en el nivel Low, donde los intentos automatizados permitieron repetir la prueba de forma controlada y comparar las respuestas del servidor frente a diferentes credenciales. En el nivel Impossible, la ejecución automatizada requirió una solicitud más específica mediante curl, incorporando la cookie de sesión y el valor security=impossible, como se presenta en la Ilustración 21.

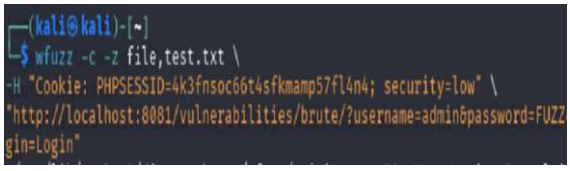

Ilustración 20
Comando Wfuzz utilizado para Automatizar Intentos de Autenticación en DVWA Nivel Low

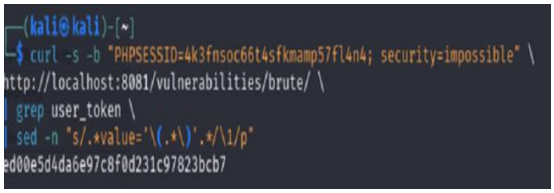

Ilustración 21
Comando Curl utilizado para Automatizar Intentos de Autenticación en DVWA Nivel Impossible

En el nivel Low, todas las respuestas devolvieron el código 200 OK, por lo que el código HTTP por sí solo no fue suficiente para distinguir entre intentos fallidos y exitosos. La Ilustración 22 presenta los resultados de Wfuzz durante esta prueba, donde el tamaño de respuesta permitió identificar diferencias entre las credenciales probadas. Los intentos fallidos mostraron un valor aproximado de 4375 Ch, mientras que el payload password produjo una respuesta de 4413 Ch, lo que indicó una variación asociada al intento exitoso.

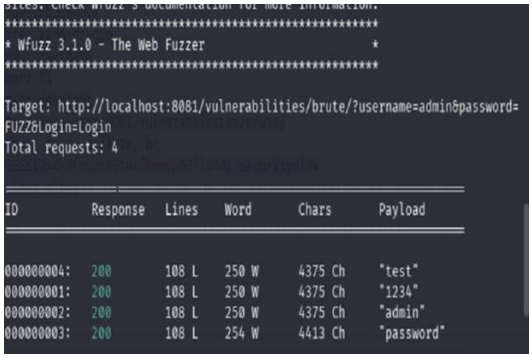

Ilustración 22
Resultados de Wfuzz en DVWA Nivel Low durante la Prueba de Fuerza Bruta

En el nivel Impossible, cada intento requirió extraer un user_token válido antes de enviar la solicitud de autenticación. La Ilustración 23 documenta esta ejecución mediante curl, donde se observa que el script obtiene el token y luego realiza los intentos contra el módulo Brute Force de DVWA manteniendo la cookie de sesión y el valor security=impossible. A diferencia del nivel Low, donde el tamaño de respuesta permitió distinguir el intento correcto, en Impossible todos los intentos devolvieron el mismo tamaño de respuesta, 4636, incluyendo el payload password. Este resultado evidencia que la aplicación aplicó controles más fuertes y redujo la posibilidad de identificar credenciales válidas mediante diferencias visibles en la respuesta del servidor.

Ilustración 23
Resultados Automatizados con Curl en DVWA Nivel Impossible durante la Prueba de Fuerza Bruta

Inyección de Comandos

La prueba comenzó en la funcionalidad DNS Lookup de OWASP Mutillidae II, donde se ingresó un valor en el campo Hostname/IP desde la aplicación web mostrándose en la Ilustración 24. La solicitud generada fue interceptada con Burp Suite para revisar el parámetro target_host, la cookie de sesión PHPSESSID y el comportamiento inicial del servidor. A partir de esa solicitud se continuó la validación manual y automatizada para determinar si la aplicación procesaba únicamente una consulta DNS normal o si permitía ejecutar comandos adicionales en el servidor.

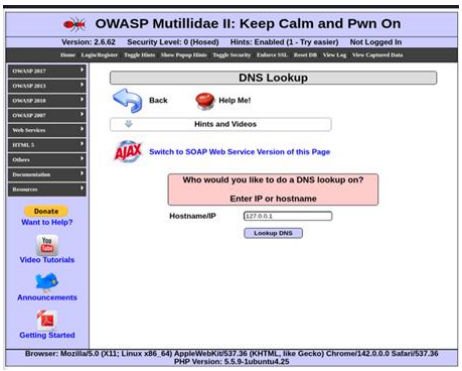

Ilustración 24
Funcionalidad DNS Lookup de OWASP Mutillidae II utilizada para la Prueba de Command Injection

En Security Level 0, la solicitud fue reproducida con curl utilizando la cookie de sesión capturada previamente. La Ilustración 25 documenta la

modificación del parámetro target_host con el payload 127.0.0.1;whoami, lo que permitió evaluar si la aplicación procesaba únicamente la consulta DNS o si también ejecutaba instrucciones adicionales en el servidor. La respuesta incluyó el valor www-data, confirmando que el comando fue ejecutado en el contexto del servidor web. Este resultado evidenció una exposición crítica en el nivel de seguridad más bajo, ya que la entrada del usuario fue interpretada como parte de un comando del sistema.

Ilustración 25
Ejecución del Payload 127.0.0.1; whoami en OWASP Mutillidae II Security Level 0

En Security Level 5, se repitió la solicitud con la cookie de sesión correspondiente y el mismo payload 127.0.0.1;whoami. La aplicación devolvió Invalid Input, lo que confirmó que los controles de validación del nivel más restrictivo bloquearon la ejecución del comando, Ilustración 26.

Ilustración 26
Bloqueo del Payload 127.0.0.1; whoami en OWASP Mutillidae II Security Level 5

Conclusiones y Trabajo Futuro

Este proyecto permitió evaluar vulnerabilidades comunes en aplicaciones web dentro de un entorno controlado, utilizando OWASP Juice Shop, DVWA y OWASP Mutillidae II. Los resultados mostraron que fallas como inyección SQL, XSS, fuerza bruta, inyección de comandos, exposición de directorios y ausencia de encabezados de seguridad pueden comportarse de manera distinta según la arquitectura, configuración y nivel de seguridad de cada aplicación. También se confirmó que combinar herramientas manuales y automatizadas fue necesario para interpretar mejor los hallazgos, ya que no todos los resultados podían entenderse únicamente por una alerta automática o por el código HTTP recibido.

Las pruebas demostraron que aplicar controles defensivos puede reducir parte de la exposición observada, pero no siempre elimina la causa principal de una vulnerabilidad. En varios casos, las medidas aplicadas ayudaron a limitar ciertos intentos o mejorar la respuesta de la aplicación, pero la seguridad real también depende de cómo se validan las entradas, cómo se manejan las sesiones, cómo se procesan las consultas y cómo está construida la lógica interna del sistema. Por eso, una defensa efectiva no debe depender de un solo control, sino de varias capas que trabajen juntas.

Como trabajo futuro, sería útil aplicar estas estrategias en una aplicación web propia o en un sistema donde se tenga autorización completa para modificar el código y la configuración. En este proyecto no se utilizaron páginas web públicas porque realizar pruebas de seguridad sin permiso puede traer problemas legales, éticos y operacionales. Una continuación natural sería desarrollar dos versiones de una misma aplicación: una sin mitigaciones y otra protegida con controles defensivos completos. Esto permitiría comparar con mayor claridad cómo responde una aplicación vulnerable frente a una aplicación reforzada, con la ventaja de poder modificar, corregir y volver a evaluar cada componente sin limitaciones externas.

Referencias

[1] I. Chorell and C. Ekberg, “A Comparative Analysis of Open Source Dynamic Application Security Testing Tools,” Linköping University, 2024.

[2] E. A. Altulaihan, A. Alismail, and M. Frikha, “A Survey on Web Application Penetration Testing,” Electronics, 2023, vol. 12, no. 5, p. 1229.

[3] X. Wang, J. Zhai, and H. Yang, “Detecting Command Injection Attacks in Web Applications Based on Novel Deep Learning Methods,” Scientific Reports, 2024, vol. 14.

[4] U. Kishnani and S. Das, “Securing the Web: Analysis of HTTP Security Headers in Popular Global Websites,” 2024.